

DR. RUPRECHT

KOMMENTIERTES ROM-LISTING



ECEC/BOOT-ROM/DOS-BOOT

Dr. Ruprecht

Kommentiertes ROM-LISTING für Commodore Amiga

Band 1

Impressum:

Herausgegeben als Gemeinschaftsproduktion der Firmen:

EDOTRONIK, BIOSYSTEMS, und GIV, München

Redaktionelle Bearbeitung: Edotronik GmbH · München
Vertrieb: BIOSYSTEMS SRI GmbH · München
für den Buchhandel: Mediscript-Verlag · München

Druck: Treiber-Offset · München
Umschlaggestaltung: Grafik-Design W. Schmid · München

ISBN 3-88320-168-5

Inhaltsverzeichnis

Das Amiga-Betriebssystem Exec 7

1 Organisation der Daten 7

2 Speicherverwaltung 12

3 Interrupt-Verwaltung 16

4 Tasks 20

5 Multitasking 24

6 Kommunikation zwischen den Tasks 25

7 Ausnahme-Zustände 31

8 Bibliotheken (Libraries) 33

9 Gerätetreiber (Devices), Ein-/Ausgabe 37

10 Spezielle Libraries für Hardware-Zugriffe (Resources) 40

11 Weitere Exec-Routinen 40

12 Der Systemstart 42

13 Die Listings 45

Exec - Sprungliste 33 . 192 51

Exec - Library (RAM - BASE) 55

Tabelle der residenten Moduln 63

Abkürzungen 65

Amiga Exec 33 . 192 67

DOS - Bootstrap 197

DOS - Bootblock 211

Amiga . Bootrom 213

Das Amiga-Betriebssystem Exec

1 Organisation der Daten

Im AMIGA-Betriebssystem gibt es (neben den speziellen Hardware-Registern) nur eine einzige feste Adresse: 4.

Alle Systemzugriffe auf Datenstrukturen gehen im Prinzip von dieser Adresse aus. Der Vorteil dieses Verfahrens ist, daß die Entwicklung des Systems keinerlei Einschränkungen durch feste Tabellen usw. unterliegt; der Preis für diesen Vorteil ist ein erhöhter Zeitaufwand für die Systemroutinen, was einen entsprechend schnellen Prozessor voraussetzt.

In den Speicherzellen 4 bis 7 steht die Basisadresse der Exec-Library, die auch als SysBase bezeichnet wird. Diese Adresse wird bei der Systeminitialisierung in das Register A6 übernommen. Alle Zugriffe auf die Exec-Routinen und den Exec-Datenbereich erfolgen dann relativ zu dieser Adresse mittels der Adressierung 'Offset(A6)', und zwar wird mit negativen Offsets auf die Exec-Sprungliste und mit positiven Offsets auf die Exec-Daten zugegriffen. (Diese Anordnung sichert die Aufwärtskompatibilität: Sowohl die Sprungliste als auch der Datenbereich können erweitert werden, ohne daß sich an den alten Offsets etwas ändert.)

In der hier beschriebenen Exec-Version gelten folgende Eckdaten:

	ohne FAST MEMORY	mit FAST MEMORY
SysBase	\$676	\$C00276
Exec-Sprungliste	\$400 - \$675	\$C00000 - \$C00275
Exec-Datenbereich	\$676 - \$8C1	\$C00276 - \$C004C1

In diesem kleinen Bereich können natürlich nicht alle Daten untergebracht werden, die Exec für seine umfangreichen Aufgaben benötigt. Die Menge dieser Daten liegt auch nicht fest, sondern ist dynamisch veränderlich je nach der Zahl und Komplexität der Programme (Tasks), die gleichzeitig verwaltet werden müssen. Der Zugriff auf diese vielfältigen Datenstrukturen erfolgt über sogenannte Listen (Lists), die die Strukturen miteinander und mit dem Listenkopf (List Header) verbinden. Dabei werden Strukturen gleicher Art jeweils in einer eigenen Liste zusammengefaßt; so gibt es z.B. eine Speicherliste (Free List) für die freien Speicherbereiche, eine Programm-Warteliste (TaskWait List) für Programme, die den Prozessor im Moment nicht benötigen, weil sie auf eine Eingabe oder ein anderes Ereignis warten, eine Programmbibliotheksliste (Library List), in der alle verfügbaren Bibliotheken (Grahics Library, DOS Library u.a.) versammelt sind, und viele andere mehr. Die Zahl der Listen ist nicht beschränkt, und gelegentlich wird von Exec oder auch von einem Programm vorübergehend eine Liste angelegt, um bestimmte Daten zu organisieren.

Jede Liste besteht aus Knoten (Nodes), die am Anfang der jeweiligen Datenstruktur stehen. Ein Knoten besitzt folgende Gestalt:

```
DC.L  ln_Succ     Zeiger auf den Nachfolger-Knoten
DC.L  ln_Pred     Zeiger auf den Vorgänger-Knoten
DC.B  ln_Type     Knotentyp (Art der Datenstruktur)
DC.B  ln_Pri      Priorität (bestimmt Platz in der Liste)
DC.L  ln_Name     Zeiger auf Knotennamen
```

Folgende Knotentypen werden von der vorliegenden Exec-Version verwendet:

0	nt_Unknown	8	nt_Resource
1	nt_Task	9	nt_Library
2	nt_Interrupt	10	nt_Memory
3	nt_Device	11	nt_SoftInt
4	nt_MsgPort	12	nt_Font
5	nt_Message	13	nt_Process
6	nt_FreeMsg	14	nt_Semaphore
7	nt_ReplyMsg	15	nt_SignalSem

Der Zweck der entsprechenden Datenstrukturen wird in den folgenden Abschnitten besprochen. Die Priorität spielt nur bei wenigen Knotentypen eine Rolle und hat sonst im allgemeinen den Wert 0. Der Knotenname wird von Exec oder anderen Programmen zur Identifizierung des Knotens und der mit ihm verbundenen Datenstruktur benutzt; er muß mit Kode 0 abgeschlossen sein.

Jede Liste hat einen Anfangs- und einen Endknoten. Im Anfangsknoten ist ln_Pred = 0, im Endknoten ist ln_Succ = 0. Diese beiden Knoten werden nicht zur Kennzeichnung von Datenstrukturen benützt; sie werden vielmehr miteinander kombiniert und bilden dann den Listenkopf (List Header):

```
DC.L   lh_Head       ln_Succ im Anfangsknoten
DC.L   lh_Tail = Ø   ln_Pred im Anfangs-, ln_Succ im Endknoten
DC.L   lh_TailPred   ln_Pred im Endknoten
DC.B   lh_Type       Listentyp (= Typ der Knoten)
DC.B   lh_Pad        Füllbyte
```

Eine Liste ist durch den Typ bestimmt und hat keinen Namen. Das letzte Byte lh_Pad ist nötig, um die Länge des Headers geradzahlig zu machen. Eine leere Liste hat außer dem Header keine weiteren Knoten; daher zeigt lh_Head auf lh_Tail und lh_TailPred auf lh_Head. Wird eine Liste neu angelegt, so ist der Header in dieser Weise zu initialisieren (vgl. z.B. Listing 02BE-02C6).

Ein Knoten wird in eine Liste eingefügt, indem ln_Succ im Vorgängerknoten und ln_Pred im Nachfolgerknoten auf den einzufügenden Knoten und ln_Succ bzw. ln-Pred im einzufügenden Knoten auf den Nachfolger- bzw. den Vorgängerknoten gerichtet werden. Wo der einzufügende Knoten im Speicher steht, ist dabei völlig belanglos. Für den Zugriff auf einen bestimmten Knoten geht Exec (oder auch ein anderes Programm) immer vom List Header aus und sucht den Knoten mittels der Routine FindName.

Ein Knoten wird aus einer Liste entfernt, indem ln_Succ im Vorgängerknoten auf den Nachfolgerknoten und ln_Pred im Nachfolgerknoten auf den Vorgängerknoten gerichtet wird. Der Knoten selbst wird jedoch nicht gelöscht, der von ihm belegte Speicherplatz nicht freigegeben.

Die von Exec unmittelbar benötigten Listen sind mit ihren List Headers im Exec-Datenbereich verankert:

Offset	Typ	Bezeichnung
$142	10	Free List
$150	8	Resource List
$15E	3	Device List
$16C	2	Interrupt List
$17A	9	Library List
$188	4	Port List
$196	1	TaskReady List
$1A4	1	TaskWait List
$1B2	11	Soft Interrupt List (Priority -32)
$1C2	11	Soft Interrupt List (Priority -16)
$1D2	11	Soft Interrupt List (Priority 0)
$1E2	11	Soft Interrupt List (Priority 16)
$1F2	11	Soft Interrupt List (Priority 32)
$214	15	Semaphore List

Die Bedeutung der an den Knoten dieser Listen hängenden Datenstrukturen wird in den nachfolgenden Abschnitten erläutert.

Zur Verwaltung der Listen stellt Exec eine Reihe von Routinen zur Verfügung, die über die Exec-Sprungliste aufgerufen werden können. In der folgenden Zusammenstellung ist außer dem Offset in der Sprungliste auch die absolute Adresse angegeben. Diese Angabe dient nur dazu, die Routine im Listing schnell zu finden; die absolute Adresse sollte nie in einem Aufruf verwendet werden, da sie schon in der nächsten Exec-Version wahrscheinlich nicht mehr stimmt!

```
Routinenname  Offset  Adresse  Beschreibung
----------------------------------------------------------------------------

Insert        -$0EA   $FC15AC  Knoten in Liste einfügen
                               Eingabe:  A0 -> List Header
                                         A1 -> Einzufügender Knoten
                                         A2 -> Knoten vor der Einfügestelle

AddHead       -$0F0   $FC15D8  Knoten am Anfang der Liste einfügen
                               Eingabe:  A0 -> List Header
                                         A1 -> Einzufügender Knoten
```

AddTail	-$0F6	$FC15E8	Knoten am Ende der Liste einfügen Eingabe: A0 -> List Header A1 -> Einzufügender Knoten
Remove	-$0FC	$FC1600	Knoten aus Liste entfernen Eingabe: A1 -> Zu entfernender Knoten
RemHead	-$102	$FC160E	Knoten am Anfang der Liste entfernen Eingabe: A0 -> List Header Ausgabe: D0 -> Entfernter Knoten D0 = 0, falls Liste leer war
RemTail	-$108	$FC161E	Knoten am Ende der Liste entfernen Eingabe: A0 -> List Header Ausgabe: D0 -> Entfernter Knoten D0 = 0, falls Liste leer war
Enqueue	-$10E	$FC1634	Knoten in eine nach Prioritäten sortierte Liste einfügen, und zwar unmittelbar vor dem ersten Knoten mit niedrigerer Priorität Eingabe: A0 -> List Header A1 -> Einzufügender Knoten
FindName	-$114	$FC165A	Knoten mit vorgegebenem Namen suchen Eingabe: A0 -> List Header oder letzter Knoten vor Suchbeginn A1 -> Name mit Endekode 0 Ausgabe: D0 -> Knoten mit vorgegebenem Namen D0 = 0, falls nicht gefunden

2 Speicherverwaltung

Der Schreib-Lese-Speicher (RAM) des AMIGA zerfällt in zwei grundsätzlich verschiedene Teile:

a) das CHIP MEMORY liegt in den untersten 512 kBytes des Adreßraumes, umfaßt also die Adressen $0 - $7FFFF (0 - 524287). Die Spezialchips, die für Graphik, Ton, Disk-Kontrolle u.a. zuständig sind, können nur auf diesen Bereich zugreifen.

b) Als FAST MEMORY wird jede über den Bereich des CHIP MEMORY hinausgehende Speichererweiterung bezeichnet, und zwar deshalb, weil der Hauptprozessor 68000 auf diesen Bereich ohne Konkurrenz durch andere Bausteine und daher ungebremst zugreifen kann.

In der Grundausstattung verfügt der AMIGA 1000 nur über das CHIP MEMORY; durch die vorn im Gehäuse einzusteckende Speicherkassette wird das CHIP MEMORY um 256 kByte auf seine Maximalgröße erweitert.

Der Adreßbereich $C00000 - $DBFFFF wird bei der Initialisierung von Exec auf das Vorhandensein von RAM-Chips geprüft und gegebenenfalls als FAST MEMORY in die Speicherverwaltung einbezogen. Falls in diesem oberen Bereich RAM-Speicher gefunden wird, legt Exec seinen Datenbereich dort an. Dies ist im Amiga 2000 realisiert: Dort sind 512 kByte FAST MEMORY im Bereich $C00000 bis $C7FFFF eingebaut.

Zu jedem der beiden RAM-Bereiche, CHIP MEMORY und FAST MEMORY, gehört ein MEMORY REGION HEADER (MRH); ist kein FAST MEMORY vorhanden, so gibt es nur den MRH des CHIP MEMORY. Dieser steht dann unmittelbar hinter dem Exec-Datenbereich, wo der freie Teil des CHIP MEMORY beginnt (Offset $24C). Jeder MRH enthält am Anfang einen Knoten. Diese Knoten gehören zur FREE LIST, deren Header im Exec-Datenbereich bei Offset $142 steht.

Ein MRH hat folgende Struktur:

```
DC.L   mh_Succ         Zeiger auf nächsten MRH
DC.L   mh_Pred         Zeiger auf vorhergehenden MRH
DC.B   mh_Type         Knotentyp = 10
DC.B   mh_Pri          Priorität (CHIP: -10, FAST: 0)
DC.L   mh_Name         Zeiger auf Namen (Chip Memory / Fast Memory
DC.W   mh_Attributes   Speichertyp
DC.L   mh_First        Zeiger auf ersten freien Block (Chunk)
DC.L   mh_Lower        Zeiger auf Anfang der Region
DC.L   mh_Upper        Zeiger auf Ende der Region
DC.L   mh_Free         Gesamtzahl der freien Bytes in der Region
```

Der Speichertyp (mh_Attributes) wird aus folgenden Bitwerten durch Addition kombiniert:

```
PUBLIC:     1
CHIP:       2
FAST:       4
```

Bei der Initialisierung des MRH wird mh_Upper mit der ersten Adresse hinter der Region, mh_Lower und mh_First mit der ersten durch 8 teilbaren Adresse hinter dem MRH belegt. Der Bereich von mh_First bis mh_Upper ist ein einziger großer freier Speicherblock (Chunk). Am Anfang dieses Blocks steht ein Blockvorspann (Chunk Header):

```
DC.L   mc_Next    Zeiger auf den nächsten freien Block
DC.L   mc_Bytes   Anzahl der Bytes in diesem freien Block
```

Im letzten (oder einzigen) Chunk Header ist mc_Next = 0. Bei einer Speicheranforderung durch das System wird nun der erste Block, der genügend freie Bytes enthält, vom Anfang her reserviert; dabei wird auch der Vorspann einbezogen. Unmittelbar hinter dem reservierten Teil wird ein neuer, korrigierter Vorspann angelegt, es sei denn, die Anforderung entspricht genau der Blocklänge; in diesem Fall wird, wenn es nicht der letzte Block war, einfach der nächste Vorspann korrigiert. Außerdem wird mc_Next im vorausgehenden Block (bzw. beim ersten Block mh_First) korrigiert.

Die Freigabe eines belegten Speicherblocks erfolgt in analoger Weise. Damit bei diesem Verfahren nicht ein Vorspann teilweise überschrieben werden kann, sorgt das System dafür, daß jede Speicherreservierung und -freigabe stets an einer durch 8 teilbaren Adresse beginnt und endet.

Diese Gesamt-Speicherliste gestattet es dem System zwar jederzeit festzustellen, welche Speicherzellen belegt und welche frei sind. Es geht aber nicht aus ihr hervor, von welchen Programmen die belegten Speicherblöcke reserviert wurden, denn die reservierten Bereiche haben kein "Etikett".

Es ist deshalb wichtig, daß die einzelnen Programme (Tasks) selbst dafür sorgen, daß belegter Speicher wieder freigegeben wird, wenn er nicht mehr benötigt wird, insbesondere dann, wenn das Programm seine Aufgabe erfüllt hat und aus dem Speicher entfernt wird. Zu diesem Zweck verwaltet jedes Programm eine eigene Speicherliste MEMORY LIST, deren List Header im Task-Kontrollblock (vgl. Abschnitt 4) untergebracht ist. Die zugehörigen Listenelemente sind sog. ML-Strukturen von folgender Gestalt:

```
DC.L   ml_Succ         Zeiger auf den nächsten Knoten
DC.L   ml_Pred         Zeiger auf den vorhergehenden Knoten
DC.B   ml_Type         Knotentyp = 10
DC.B   ml_Pri          Priorität (i.allg. = 0)
DC.L   ml_Name         Zeiger auf Namen oder 0
DC.W   ml_NumEntries   Zahl der folgenden Einträge (ME)
DC.L   me_Addr         Anfang des Speicherblocks
DC.L   me_Length       Länge des Speicherblocks
```

Die beiden letzten Zeilen bilden einen "Speicher-Eintrag" (Memory Entry, ME). Jede ML-Struktur kann beliebig viele solche Einträge enthalten; ihre Anzahl wird durch ml_NumEntries angegeben. Ein Programm muß also nicht einen zusammenhängenden Speicherblock reservieren lassen, was bei der beschriebenen Art der Speicherverwaltung leicht erfolglos sein kann, sondern es kann seinen Speicherbedarf auf kleinere Blöcke, u.U. in verschiedenen Regionen, aufteilen. Für die Speicheranforderung wird ebenfalls die ML-Struktur verwendet, dabei wird statt me_Addr (die Adresse wird erst durch die Reservierungsroutine geliefert) zunächst der geforderte Speichertyp me_Reqs eingetragen.

Die möglichen Typen sind - wie im MRH - PUBLIC, CHIP, FAST; außerdem kommen noch die Anforderungen CLEAR (Bit 16) und LARGEST (Bit 17) hinzu:

CLEAR bewirkt, daß der reservierte Speicherblock gelöscht (d.h. mit Kode 0 gefüllt) wird, LARGEST bewirkt, daß die Routine AvailMem (s.u.) den größten noch freien Speicherblock in der angegebenen Region ermittelt. Wird keine bestimmte Speicherregion angefordert, so versucht Exec zunächst, einen Bereich im FAST MEMORY zuzuweisen (dies ergibt sich aus der Priorität im Memory Region Header!)

Wird ein Programm auf ordentliche Weise abgeschlossen, z.B. mittels der Systemroutine RemTask, so werden alle von dem Programm belegten Speicherblöcke anhand der ML-Liste, deren List Header im Task-Kontrollblock steht, wieder freigegeben.

Für die Speicherverwaltung stellt Exec folgende Routinen zur Verfügung:

```
Routinenname  Offset  Adresse  Beschreibung
--------------------------------------------------------------------------
Allocate      -$0BA   $FC169C  Speicherblock reservieren
                               Eingabe: A0 -> Memory Region Header
                                        D0 = Blocklänge in Bytes
                               Ausgabe: D0 -> Anfang des reservierten Blocks
                                        D0 = 0, falls erfolglos

AllocMem      -$0C6   $FC1794  Speicherblock reservieren
                               Eingabe: D0 = Blocklänge in Bytes
                                        D1 = Typ (PUBLIC,CHIP,FAST,CLEAR)
                               Ausgabe: wie Allocate.

AllocEntry    -$0DE   $FC191E  Speicherblöcke gemäß ML-Struktur reservieren
                               Eingabe: A0 -> ML-Struktur (Anforderung)
                               Ausgabe: D0 -> ML-Struktur (Reservierung)
                                        D0 = Typ mit Bit 31 gesetzt, falls
                                             erfolglos; in diesem Fall wird
                                             kein Speicher belegt.

AllocAbs      -$0CC   $FC1840  Speicherblock ab gegebener Adresse reservieren
                               Eingabe: A1 -> Anfang des Speicherblocks
                                        D0 = Blocklänge in Bytes
                               Ausgabe: D0 -> Anfang des reservierten Blocks
                                        D0 = 0, falls erfolglos
```

Deallocate	-$0C0	$FC1704	Speicherblock freigeben
			Eingabe: A0 -> Memory Region Header
			A1 -> Anfang des Speicherblocks
			D0 = Blocklänge in Bytes
FreeMem	-$0D2	$FC17F0	Speicherblock freigeben
			Eingabe: A1 -> Anfang des Speicherblocks
			D0 = Blocklänge in Bytes
FreeEntry	-$0E4	$FC19AC	Speicherblöcke gemäß ML-Struktur freigeben
			Eingabe: A0 -> ML-Struktur
AvailMem	-$0D8	$FC18D0	Freien Speicherplatz ermitteln
			Eingabe: D1 = Typ (PUBLIC,CHIP,FAST,LARGEST)
			Ausgabe: D0 = Freier Speicherplatz
			(bei LARGEST: im größten zusammen-
			hängenden Block; sonst: insgesamt)
TypeOfMem	-$216	$FC181A	Speichertyp ermitteln
			Eingabe: A1 -> Anfang eines Speicherblocks
			Ausgabe: D0 = Speichertyp (aus MRH)
			D0 = 0, wenn nicht gefunden
AddMemList	-$26A	$FC19EA	Memory Region Header zur Free List hinzufügen
			Eingabe: A0 -> Anfang der Speicherregion
			A1 -> Name der Region oder = 0
			D0 = Größe der Region in Bytes
			D1 = Speichertyp
			D2 = Priorität

3 Interrupt-Verwaltung

Unterbrechungen (Interrupts) spielen in einem Multitasking-System eine besondere Rolle. Es ist daher notwendig, sich mit der Interrupt-Verarbeitung durch den Prozessor 68000 und das Betriebssystem Exec vertraut zu machen.

Der Prozessor hat 3 Eingänge für externe Interrupts, denen drei Bits im Statusregister entsprechen. Sowohl der Zustand der Eingänge als auch die Statusbits lassen sich als Binärzahl aus dem Bereich 0 bis 7 darstellen. Jeder Zahl entspricht eine Prioritätse-

bene im folgenden Sinn: Wenn ein Interrupt-Signal an einer Kombination der Eingänge empfangen wird, die der Ebene i entspricht, so wird diese Unterbrechung vom Prozessor dann und nur dann zugelassen, wenn die Interrupt-Bits im Statusregister eine Zahl bilden, die kleiner ist als i. In diesem Fall werden die Interrupt-Bits vom Prozessor auf den Wert i gesetzt mit der Folge, daß während der Bearbeitung einer Unterbrechung nur dann eine weitere Unterbrechung angenommen wird, wenn diese eine höhere Priorität besitzt. Die Ebene 0 bedeutet, daß keine Unterbrechung angefordert, aber jede zugelassen ist; die Ebene 7 stellt insofern einen Sonderfall dar, als sie auch durch den Wert 7 der Statusbits nicht gesperrt werden kann: Ein Interrupt-Signal an allen 3 Eingängen erzeugt einen "nicht maskierbaren Interrupt" (NMI). Die Bearbeitung der Unterbrechung durch den Prozessor beginnt damit, daß aus der Interrupt-Ebene die Adresse eines Interrupt-Vektors berechnet wird, der auf die Bearbeitungsroutine zeigt. Die 7 Interrupt-Vektoren stehen an folgende Adressen und haben (in dieser Exec-Version) folgende Werte:

Ebene	Adresse	Vektor
1	$Ø64	$FCØC52
2	$Ø68	$FCØCA6
3	$Ø6C	$FCØCD8
4	$Ø7Ø	$FCØD3Ø
5	$Ø74	$FCØDBE
6	$Ø78	$FCØEØ4
7	$Ø7C	$FCØE4A

Diese kurze Beschreibung der Interrupt-Verarbeitung durch den Prozessor 68000 kann hier genügen; genauere Einzelheiten können einem einschlägigen Prozessor-Handbuch entnommen werden. Für Exec reichen die 7 Interrupt-Ebenen des Prozessors nicht aus; sie werden deshalb gemäß folgender Tabelle auf 16 Ebenen ausgedehnt:

Priorität Exec	CPU	Name	IV Ofs.	Auslöser der Unterbrechung
0	1	TBE	$054	Serieller Port: Sendepuffer leer
1	1	DskBlk	$060	Lesen/Schreiben eines Disk-Blocks fertig
2	1	SoftInt	$06C	Vom Programm ausgelöste Unterbrechung
3	2	Ports	$078	I/O oder Zeitgeber (Baustein 6520A)
4	3	Coper	$084	Spezialchip Copper
5	3	VertB	$090	Bildwechsel-Impuls
6	3	Blit	$09C	Spezialchip Blitter fertig
7	4	Aud2	$0C0	Audio-Kanal 2
8	4	Aud0	$0A8	Audio-Kanal 0
9	4	Aud3	$0CC	Audio-Kanal 3
10	4	Aud1	$0B4	Audio-Kanal 1
11	5	RBF	$0D8	Serieller Port: Empfangspuffer voll
12	5	DskSync	$0E4	Vorgegebenes Byte von Disk gelesen
13	6	Exter	$0F0	Extern, Baustein 8520B
14	6	Inten	$0FC	Spezial (Copper)
15	7	NMI	$108	Nicht maskierbarer Interrupt

Die Interrupts können von Exec einzeln zugelassen oder gesperrt werden durch Setzen oder Löschen des entsprechenden Bits im INTENA-Register (Adresse $DFF09A). Außerdem können durch Löschen bzw. Setzen des Bits 14 in diesem Register alle Interrupts, die einzeln zugelassen sind, gemeinsam gesperrt bzw. freigegeben werden. Wenn eine der oben genannten Quellen einen Interrupt anfordert, setzt sie das entsprechende Bit im INTREQ- Register (Adresse $DFF09C); in der Verteilungsroutine (Interrupt Handler), auf die der zur Prioritätsebene gehörende Interruptvektor zeigt, prüft Exec durch Lesen des INTREQR-Registers (Adresse $DFF01E), welche Quelle innerhalb dieser Ebene die Unterbrechung ausgelöst hat, und verzweigt dann in die entsprechende Bearbeitungsroutine (Interrupt Server). Die Parameter für diese Verzweigung werden aus einer Tabelle von Interrupt-Vektoren im Exec-Datenbereich (die nicht mit den oben genannten Vektoren für die Prozessor-Interrupts verwechselt werden dürfen) entnommen; für jede der 15 Interrupt-Arten enthält diese Tabelle 3 Adressen: iv_Data zeigt auf einen Bereich mit Daten, iv_Code zeigt auf die Bearbeitungsroutine und iv_Node auf einen Knoten. Die Offsets der Tabelleneinträge im Exec - Datenbereich sind in der obigen Tabelle unter IV Ofs angegeben. Es ist zu bemerken, daß diese Tabelle nicht für alle Arten von Interrupts bei der Exec-Initialisierung bereits eingerichtet wird; vielmehr werden zunächst nur die Interrupt-Server für die Interrupts 3, 4, 5, 13 und 15 in folgender

Weise initialisiert: Es wird je ein List Header für eine Liste von Interrupt Servers erzeugt, dann wird iv_Data auf den List Header und iv_Code auf eine spezielle Interrupt-Routine gerichtet (Adresse $FC12FC), die von dem jeweiligen List Header ausgehend die einzelnen Interrupt Server in der Liste nacheinander aufruft.

Manche Systemroutinen dürfen nicht unterbrochen werden. Von solchen Routinen müssen deshalb die Unterbrechungen durch entsprechendes Schreiben in das INTENA-Register gesperrt und später wieder freigegeben werden. Hier ist jedoch ein zusätzlicher Mechanismus erforderlich, der eine vorzeitige Freigabe verhindert: Wenn eine Routine die Interrupts sperrt und dann eine andere Routine aufruft, die ebenfalls die Interrupts sperrt, so darf die zweite Routine die Interrupts nicht wieder freigeben, denn sie müssen ja bis zum Ende der ersten Routine gesperrt bleiben! Eine solche "Verschachtelung" kann über mehrere Ebenen gehen, dabei muß in jedem Fall dafür gesorgt werden, daß die Freigabe der Unterbrechungen nur durch die Routine vorgenommen werden kann, die sie als erste gesperrt hat. Dies geschieht in der Weise, daß bei jeder Sperrung ein Zähler, dessen Anfangswert -1 ist, um 1 erhöht und bei jeder Freigabe um 1 erniedrigt wird. Erst dann, wenn der Zählerinhalt wieder -1 ist, werden die Interrupts durch Schreiben in das INTENA-Register tatsächlich wieder freigegeben. Der Zähler heißt IDNestCnt (Interrupt Disable Nesting Counter = Unterbrechungssperren-Verschachtelungszähler), ist 1 Byte lang und hat im Exec-Datenbereich den Offset $126.

Eine besondere Art von Unterbrechung stellen die Software Interrupts dar: Sie werden nicht von einem externen Signal, sondern vom Programm selbst ausgelöst, indem das Bit 3 des INTREQ-Registers gesetzt wird. Dies erzeugt einen Interrupt der Priorität 1, dessen Abwicklung dann wie oben beschrieben abläuft. Voraussetzung ist, daß vorher ein entsprechender Interrupt Server bereitgestellt wurde und in eine der 5 SoftInt-Listen, deren List Header im Exec-Datenbereich ab Offset $1B2 stehen, ein Knoten eingefügt wurde, an dem ein iv_Data- und ein iv_Code-Vektor hängt. Der iv_Code-Vektor muß auf den Interrupt Server gerichtet sein.

Das Betriebssystem stellt folgende Routinen zur Interrupt-Verwaltung bereit:

```
Routinenname  Offset  Adresse  Beschreibung
-------------------------------------------------------------------------
SetIntVector  -$ØA2   $FC118E  System-Interruptvektor ändern
                               Eingabe: DØ = Int-Nummer (Ø,...,15)
                                        A1 -> Knoten mit iv_Data u. iv_Code
                               Ausgabe: DØ -> alter Knoten
```

```
AddIntServer  -$ØA8   $FC11D4  Interrupt Server Routine hinzufügen
                               Eingabe: DØ = Int-Nummer (3,4,5,13,15)
                                        A1 -> Knoten mit iv_Data u. iv_Code

RemIntServer  -$ØAE   $FC1214  Interrupt Server Routine entfernen
                               Eingabe: DØ = Int-Nummer
                                        A1 -> Knoten des Interrupt Servers

Cause         -$ØB4   $FC132Ø  Software-Interrupt erzeugen
                               Eingabe: A1 -> Knoten des Interrupt Servers

Disable       -$Ø78   $FC13EC  Interrupts sperren und IDNestCnt erhöhen


Enable        -$Ø7E   $FC13FA  IDNestCnt erniedrigen und Interrupts frei-
                               geben, falls IDNestCnt = -1
```

4 Tasks

Als Task wird ein Programm mit den zugehörigen Datenbereichen bezeichnet, das eine vom Anwender definierte Aufgabe bearbeitet. Alle Daten, die Exec zur Verwaltung der Task benötigt, sind im Task-Kontrollblock (TC) zusammengefaßt:

```
      struct  tc_Node          Knoten
14    DC.B    tc_Flags         Verschiedene Flagbits
      DC.B    tc_State         Status (Added,Run,Ready,Wait,Except,Removed)
      DC.B    tc_IDNestCnt     Int-Disable Verschachtelungsebene
      DC.B    tc_TDNestCnt     Taskwechsel-Sperre Verschachtelungsebene
18    DC.L    tc_SigAlloc      Zugeordnete Signale
      DC.L    tc_SigWait       Erwartete Signale
      DC.L    tc_SigRecvd      Empfangene Signale
30    DC.L    tc_SigExcept     Ausnahme-Signale
      DC.W    tc_TrapAlloc     Zugeordnete Traps
      DC.W    tc_TrapAble      Freigegebene Traps
38    DC.L    tc_ExceptData    Zeiger auf Daten für Ausnahmebehandlung
42    DC.L    tc_ExceptCode    Zeiger auf Ausnahmeroutine
      DC.L    tc_TrapData      Zeiger auf Trap-Daten
      DC.L    tc_TrapCode      Zeiger auf Trap-Routine
```

DC.L	tc_SPReg	Stack-Zeiger
DC.L	tc_SPLower	Zeiger auf Stack-Untergrenze
DC.L	tc_SPUpper	Zeiger auf Stack-Obergrenze
DC.L	tc_Switch	Zeiger auf Abgaberoutine
DC.L	tc_Launch	Zeiger auf Übernahmeroutine
struct	tc_MemEntry	List Header für Task-Speicherliste
DC.L	tc_UserData	Zeiger auf Task-Daten

Die Einträge im Task-Kontrollblock haben folgende Bedeutung:

tc_Node ist ein Knoten (vgl. Abschnitt 1), mit dem die Task in eine von zwei Listen eingereiht werden kann: Die Ready-Liste enthält alle Tasks, nach Priorität geordnet, die bereit sind, den Prozessor zu übernehmen, die Wait-Liste enthält diejenigen Tasks, die zur Zeit "schlafen", weil sie auf ein Signal von einer anderen Task, z.B. einem Gerätetreiber, warten.

tc_Flags enthält verschiedene Flagbits, von denen Exec zur Zeit nur folgende benützt:

Bit 5	tb_Except	Task bearbeitet Ausnahme
Bit 6	tb_Switch	tc_Switch zeigt auf Switch-Routine (Aufruf bei Abgabe des Prozessors)
Bit 7	tb_Launch	tc_Launch zeigt auf Launch-Routine (Aufruf bei Übernahme des Prozessors)
Bit 0	tb_ProcTime	diese beiden Flagbits sind für
Bit 4	tb_StackChk	künftige Erweiterungen reserviert

tc_State reflektiert den Task-Zustand und kann folgende Werte annehmen:

1	ts_Added	Task wurde dem System übergeben
2	ts_Run	Task ist aktiv
3	ts_Ready	Task bereit, aber nicht aktiv
4	ts_Wait	Task wartet
5	ts_Except	Task bearbeitet Ausnahme
6	ts_Removed	Task nicht in Ready- oder Wait-Liste

tc_IDNestCnt und tc_TDNestCnt speichern die aktuellen Werte der Verschachtelungszähler, wenn die Task in den Wartezustand geht; in diesem Fall werden von der Routine WAIT Interrupts und Taskwechsel freigegeben, damit die Zeit von anderen Tasks genutzt werden kann. Sobald das erwartete Signal eintrifft und die Task wieder "geweckt" wird, stellt Exec den ursprünglichen Wert der Verschachtelungszähler wieder her.

tc_SigAlloc enthält 32 Signalbits, die bestimmten Ereignissen zugeordnet werden können und bestimmte Reaktionen der Task auslösen. Bits Ø bis 15 sind für Systemzwecke reserviert, Bits 16 bis 31 können vom Anwender beliebig definiert werden.

tc_SigWait enthält eine Maske derjenigen Signale, auf die die Task gerade wartet.

tc_SigRecvd Wenn die Task ein Signal empfängt, wird das entsprechende Bit in diesem Doppelwort gesetzt.

tc_SigExcept enthält diejenigen Signalbits, die bei Empfang eine Ausnahmeverarbeitung auslösen (vgl. Abschnitt 6).

tc_TrapAlloc und tc_TrapAble unterstützen die Behandlung von Prozessor-Ausnahmen durch die Task (vgl. Abschnitt 6).

tc_ExceptData und tc_ExceptCode zeigen auf einen Datenbereich und eine Routine, die für die Verarbeitung einer Task-Ausnahme benötigt werden.

tc_TrapData und tc_TrapCode zeigen auf einen Datenbereich und eine Routine, die für die Verarbeitung einer Prozessor-Ausnahme benötigt werden.

tc_SPReg dient zur Aufbewahrung des Task-Stackzeigers beim Taskwechsel. Bei der Einrichtung der Task ist diese Variable mit dem Anfangswert des Stackzeigers zu initialisieren

tc_SPLower und tc_SPUpper enthalten die Unter- und Obergrenze des Task-Stacks. Die Werte sind bei der Initialisierung einzutragen.

tc_Switch und tc_Launch dienen dazu, besondere Routinen für die Abgabe bzw. Übernahme des Prozessors beim Taskwechsel aufzurufen. Dies geschieht jedoch nur, wenn die entsprechenden Bits in tc_Flags gesetzt sind.

tc_MemEntry ist der List Header der Speicherliste, in der die von der Task belegten Speicherbereiche zusammengefaßt sind. Wegen der Einzelheiten wird auf Abschnitt 2 verwiesen.

tc_UserData wird vom System nicht benützt und kann von der Task selbst nach Belieben verwendet werden.

Für das Hinzufügen, Entfernen und Suchen einer Task stellt Exec folgende Routinen bereit:

Routinenname	Offset	Adresse	Beschreibung
AddTask	-$11A	$FC1C48	Eine Task dem System hinzufügen Eingabe: A1 -> Task-Kontrollblock A2 -> Programm-Anfang A3 -> Abschluß-Routine
RemTask	-$120	$FC1CF4	Task aus dem System entfernen Eingabe: A1 -> Task-Kontrollblock
FindTask	-$126	$FC1D74	Task suchen Eingabe: A1 -> Name der gesuchten Task A1 = 0: Laufende Task suchen Ausgabe: D0 -> Task-Kontrollblock D0 = 0: Task nicht gefunden

5 Multitasking

Multitasking bedeutet, daß im Amiga mehrere Tasks "gleichzeitig" laufen können. Dabei ist "gleichzeitig" nicht wörtlich zu nehmen: Der Prozessor kann immer nur an einer Task arbeiten, er kann aber im raschen zeitlichen Wechsel mehrere Tasks bedienen. Wenn eine Task den Prozessor verliert, werden alle Prozessor-Register im Task-Stack und alle sonstigen wichtigen Daten im Task-Kontrollblock abgespeichert; wenn der Prozessor der Task wieder zur Verfügung steht, nimmt er die Arbeit genau an der Stelle wieder auf, wo er sie abgebrochen hat. Der Taskwechsel geschieht in aller Regel während eines Interrupts: Alle Interrupt-Verteilerroutinen (Interrupt Handler) für die Prioritätsebenen 1 bis 6 münden in die Routine ExitIntr. Dort wird geprüft, ob ein Taskwechsel ansteht und ob er zugelassen ist. Ist dies der Fall, so wird die Routine Schedule aufgerufen; diese prüft, ob eine der folgenden Voraussetzungen erfüllt ist:

1. eine Task mit gleicher oder höherer Priorität wurde in die Ready-Liste eingefügt,
2. eine Task mit gleicher Priorität wartet in der Ready-Liste auf den Prozessor und die laufende Task hat ihr Zeitquantum verbraucht.

Ist dies der Fall, so wird die Routine Switch aufgerufen. Sie beendet die Arbeit des Prozessors an der laufenden Task, indem sie alle wichtigen Daten im Task-Kontrollblock abspeichert und, falls eine besondere Routine zu bearbeiten ist, auf die tc-Switch zeigt, diese aufruft. Anschließend wird die Routine Dispatch gerufen, die die nächste Task an den Prozessor übergibt. Ein Taskwechsel kann von der laufenden Task selbst ausgelöst werden, wenn sie auf ein Signal von einer anderen Task warten muß. Dies geschieht durch Aufruf der Routine Wait. Diese Routine stellt die laufende Task in die Warteliste und bleibt in einer Warteschleife, bis das erwartete Signal eintrifft. Bei jedem Umlauf in der Warteschleife wird die Routine Switch gerufen, die einen Taskwechsel auslöst. Eine Task mit niedriger Priorität erhält nur dann Prozessorzeit zugewiesen, wenn alle Tasks mit höherer Priorität in der Warteliste stehen.

Falls überhaupt keine Task zur Bearbeitung ansteht, wird der Prozessor von der Routine Dispatch angehalten; er stellt dann die Arbeit ein, bis er durch einen Interrupt wieder "geweckt" wird.

Manche Routinen dürfen zwar durch einen Interrupt, aber nicht durch einen Taskwechsel unterbrochen werden; deshalb kann der Taskwechsel gesperrt werden. Hier gelten ganz entsprechende Überlegungen wie bei der Sperrung von Interrupts: Die Freigabe darf nur durch diejenige Routine erfolgen, die als erste gesperrt hat. Das Verfahren ist hier das gleich wie dort: Bei jeder Sperre wird der Zähler TDNestCnt (Task Disable Nesting Counter = Taskwechselsperre-Verschachtelungszähler) um 1

erhöht, bei jeder Freigabe um 1 erniedrigt. Erst wenn der Zähler wieder im Ausgangszustand -1 angekommen ist, wird der Taskwechsel wirklich wieder freigegeben. Der Zähler ist 1 Byte lang und befindet sich im Exec-Datenbereich, Offset $127.

Die folgende Übersicht enthält die Exec-Routinen für das Multitasking:

Routinenname	Offset	Adresse	Beschreibung
ExitIntr	-$Ø24	$FCØE6Ø	Abschluß der Interrupt-Handler
Schedule	-$Ø2A	$FCØE86	Prüfen, ob Taskwechsel notwendig
Switch	-$Ø36	$FCØEEØ	Laufende Task vom Prozessor abhängen
Dispatch	-$Ø3C	$FCØF2A	Neue Task in Lauf setzen
Reschedule	-$Ø3Ø	$FC1F38	Taskwechsel durch SoftInt initiieren
Wait	-$13E	$FC1EDØ	Laufende Task in Wartezustand versetzen
SetTaskPri	-$12C	$FC1DC8	Task-Priorität ändern, Reschedule Eingabe: A1 -> Task-Kontrollblock DØ = Priorität Ausgabe: DØ = alte Priorität
Forbid	-$Ø84	$FC1F5A	Taskwechsel sperren, TDNestCnt erhöhen
Permit	-$Ø8A	$FC1F6Ø	TDNestCnt erniedrigen, Taskwechsel freigeben, falls TDNestCnt = -1

6 Kommunikation zwischen den Tasks

Es kommt sehr häufig vor, daß zwei Tasks miteinander kommunizieren müssen, weil sie gemeinsam an einer Aufgabe arbeiten. Wenn z.B. eine Task einen Text auf der Diskette speichern will, so wickelt sie den Verkehr mit dem Laufwerk nicht selbst ab, sondern ruft eine zu diesem Zweck besonders eingerichtete Task und übergibt ihr die notwendigen Anweisungen in Form von Parametern. Diese Task wiederum benachrichtigt die aufrufende Task, sobald der Auftrag ausgeführt ist. Dieser Nachrichten-

verkehr zwischen den Tasks wird über sogenannte Message Ports abgewickelt. Jeder Message Port ist ein Briefkasten, den eine Task eingerichtet hat, um darin Nachrichten empfangen zu können. Ein Message Port hat folgende Struktur:

```
struct  mp_Node         Knoten
DC.B    mp_Flags        Flags für Reaktion auf empfangene Nachricht
DC.B    mp_SigBit       Nummer des Signalbits
DC.L    mp_SigTask      Zeiger auf Task, die den Port eingerichtet hat
struct  mp_MsgList      List Header für alle empfangenen Nachrichten
```

Eine Nachricht (Message) ist eine Datenstruktur von folgender Gestalt:

```
struct  mn_Node         Knoten
DC.L    mn_ReplyPort    Zeiger auf Message Port für Antwort-Nachricht
DC.W    mn_Length       Länge des folgenden Inhalts der Nachricht
```

Der eigentliche Inhalt der Nachricht schließt sich unmittelbar an und kann eine Folge von ganz beliebigen Kodes sein, deren Bedeutung natürlich der empfangenden Task bekannt sein muß, damit sie etwas damit anfangen kann.

Der Nachrichtenaustausch funktioniert nun so: Die sendende Task konstruiert zunächst die Nachricht und ruft dann die Exec-Routine PutMsg (Put Message = Sende Nachricht), wobei sie ihr einen Zeiger auf die Nachricht und einen Zeiger auf den Port übergibt. Die Routine PutMsg hängt zuerst die Nachricht an das Ende der Liste, deren List Header im Message Port steht. Dann prüft sie, ob im Zeiger mp_SigTask etwa eine Null steht; ist dies der Fall, dann ist die Angelegenheit mit der Ablieferung der Nachricht erledigt; die empfangende Task wird schon irgendwann nachsehen, ob die Nachricht da ist. Hat der Zeiger einen von Null verschiedenen Wert, so werden nun die Aktions-Flags in mp_Flags geprüft; sie sind in den Bits 0 und 1 untergebracht und haben folgende Bedeutung:

```
Ø       pa_Signal
1       pa_SoftInt
2       pa_Ignore
```

Ist pa_Signal gesetzt, so ruft PutMsg die Routine Signal und übergibt ihr einen Zeiger auf die Task, die den Port errichtet hat, sowie die Nummer des Signalbits mp_SigBit. Diese Routine nun übergibt das Signal an die Task, vergleicht es mit den im Task-Kontrollblock gespeicherten Signalmasken und leitet die erforderlichen Reaktionen der Task ein.

Ist pa_SoftInt gesetzt, so ruft PutMsg die Routine Cause und übergibt ihr den Zeiger auf die zugehörige Interrupt-Struktur, der sich in diesem Fall an der Stelle des Zeigers mp_SigTask befindet. Die Routine Cause löst alsdann einen Software-Interrupt aus.

Ist pa_Ignore gesetzt, so tut PutMsg gar nichts.

Der Fall, daß die Aktion-Flags den Wert 3 haben, ist offiziell nicht vorgesehen; trotzdem geschieht auch hier etwas: PutMsg betrachtet den Zeiger mp_SigTask als Anfangsadresse einer Routine und ruft diese auf.

Ist die Nachricht angekommen, so wird die empfangende Task die Nachricht auswerten und bestimmte Konsequenzen aus ihr ziehen. Ist der Zweck der Nachricht erfüllt, so prüft die Task, ob eine Antwort erwartet wird; dies ist der Fall, wenn im mn_ReplyPort-Feld der Message nicht Null steht. Üblicherweise wird in einem solchen Fall die Nachricht an den ReplyPort, den die Absender-Task errichtet hat, mit entsprechenden Änderungen zurückgeschickt.

Wenn ein Message Port für eine größere Zahl von Tasks nützlich ist, so wird er zweckmäßigerweiße publik gemacht; dies geschieht durch Einfügen in die Port Liste, deren List Header im Exec-Datenbereich (Offset $188) steht. Damit der Port identifiziert werden kann, erhält er in diesem Fall einen Namen, nach dem mittels der Routine FindPort gesucht werden kann.

Die folgende Übersicht enthält die für den Umgang mit Ports, Nachrichten und Signalen verfügbaren Routinen.

Routinenname	Offset	Adresse	Beschreibung
AddPort	-$162	$FC1B18	Message Port zur System-Portliste hinzufügen Eingabe: A1 -> Port
RemPort	-$168	$FC1B30	Message Port aus System-Portliste entfernen Eingabe: A1 -> Port

```
FindPort      -$186   $FC1C1E  Message Port in System-Portliste suchen
                               Eingabe: A1 -> Name des Ports
                               Ausgabe: DØ -> Message Port
                                        DØ = Ø, falls nicht gefunden

PutMsg        -$16E   $FC1B34  Message an Message Port senden
                               Eingabe: AØ -> Message Port
                                        A1 -> Message

GetMsg        -$174   $FC1BAE  Message vom Message Port holen
                               Eingabe: AØ -> Message Port
                               Ausgabe: DØ -> Erste Message in der Liste
                                        DØ = Ø, falls Liste leer

ReplyMsg      -$17A   $FC1BDC  Message an Absender zurücksenden
                               Eingabe: A1 -> Message

AllocSignal   -$14A   $FC1FC4  Signalbit in tc_SigAlloc belegen
                               Eingabe: DØ = gewünschtes Bit (Ø,...,31)
                                             oder -1, wenn beliebig
                               Ausgabe: DØ = zugeordnetes Bit (Ø,...,31)
                                             oder -1, wenn keines frei

FreeSignal    -$15Ø   $FC1FFC  Signalbit in tc_SigAlloc freigeben
                               Eingabe: DØ = freizugebendes Bit (Ø,...,31)

SetSignal     -$132   $FC1E22  Signalbits in tc_SigRecvd setzen
                               Eingabe: DØ = neue Signalbits
                                        D1 = Signalmaske
                               Ausgabe: DØ = alte Signalbits

Signal        -$144   $FC1E48  Signal an Task senden
                               Eingabe: A1 -> Task
                                        DØ = Signal

WaitPort      -$18Ø   $FC1BF6  Auf eine Nachricht an einem Port warten
                               Eingabe: AØ -> Port
                               Ausgabe: DØ -> Erste Message am Port
```

Eine spezielle Art von Message Ports sind die Semaphores. Mit ihrer Hilfe kann eine Task den Zugriff auf eine Datenstruktur durch andere Tasks zeitweilig sperren. Dies kann notwendig sein, wenn die Daten gerade geändert werden und ein lesender oder gar schreibender Zugriff während des Änderungsvorgangs unter Umständen unsinnige Daten hervorbringen würde. Der Vorteil gegenüber der Taskwechselsperre, die dasselbe bewirkt, liegt darin, daß nur solche Tasks behindert werden, die tatsächlich auf die geschützten Daten zugreifen wollen, während im übrigen der Taskwechsel normal weiterläuft. Exec selbst verwendet Semaphores wegen des größeren Verwaltungsaufwands nicht. Für die Verwendung von Semaphores gilt folgendes: Wenn eine Task auf einen Semaphore-geschützten Datensatz zugreifen will, schickt sie mittels der Routine Procure eine Anfrage (BidMessage) an den Semaphore. Ist der Zugriff erlaubt, erhält die Task sofort eine entsprechende Rückmeldung und der Zugriff wird für weitere Tasks gesperrt. Ist der Zugriff dagegen gesperrt, so wird die BidMessage im Message Port abgelegt (mittels PutMsg) und die Task muß an ihrem ReplyPort auf die Rückmeldung warten, daß der Zugriff wieder frei ist. Ist der Zugriff auf den Datensatz beendet, so ruft die Task die Routine Vacate; diese stellt fest, ob noch andere Tasks zugreifen wollen, und sendet in diesem Fall der nächsten wartenden Task die BidMessage an deren ReplyPort zurück. Damit nicht jedesmal die Message-Liste geprüft werden muß, besteht die Semaphore-Struktur aus einem Message-Port mit einem angehängten Zähler (DC.W sm_Bids), der wie die bereits bekannten Verschachtelungszähler funktioniert: Jeder Procure-Aufruf erhöht den Zühler um 1, jeder Vacate-Aufruf erniedrigt ihn um 1; wenn keine Task wartet, steht er auf -1. Es gibt noch eine zweite Art von Semaphores, die Signal-Semaphores. Sie dienen demselben Zweck wie die vorher beschriebenen Semaphores, benützen dazu jedoch ein modifiziertes Verfahren, das keine Messages, sondern Signale zur Information der Tasks benützt. Wenn eine Task eine Zugriffsanforderung an ein Signal-Semaphore schickt und der Zugriff ist gesperrt, wird die Task in den Wartezustand versetzt, bis der Zugriff erlaubt ist. Die Task kann aber auch zunächst versuchsweise anfragen, wenn sie nicht riskieren will, daß sie warten muß. Auch diese Semaphores werden von Exec selbst nicht benützt. Ein Signal-Semaphore hat folgende Struktur:

```
struct  ss_Node          Knoten
DC.W    ss_NestCount     Zähler für Mehrfachaufruf derselben Task
struct  ss_WaitQueue     List Header für Task-Warteliste
struct  ss_MultipleLink  Zur Kopplung von Semaphores
DC.L    ss_Owner         Zeiger auf sperrende Task
DC.W    ss_QueueCount    Wartelisten-Zähler
```

Auf eine detaillierte Beschreibung des Signal-Semaphore-Verfahrens muß hier verzichtet werden; im Bedarfsfall wird das Studium der zugehörigen Routinen empfohlen, die in der folgenden Übersicht zusammengestellt sind:

```
Routinenname  Offset  Adresse  Beschreibung
---------------------------------------------------------------------------
Procure       -$21C   $FC2D5C  Zugriffs-Anforderung an Semaphore senden
                               Eingabe: A0 -> Semaphore
                                        A1 -> BidMessage
                               Ausgabe: D0 = -1, wenn Zugriff erlaubt
                                        D0 = 0, wenn Zugriff gesperrt

Vacate        -$222   $FC2D72  Semaphore wieder freigeben
                               Eingabe: A0 -> Semaphore

InitSemaphore -$22E   $FC2D94  SignalSemaphore initialisieren
                               Eingabe: A0 -> SignalSemaphore

Obtain        -$234   $FC2DB4  Zugriffsanforderung an SignalSemaphore senden
    Semaphore                  Eingabe: A0 -> SignalSemaphore

Release       -$23A   $FC2E04  SignalSemaphore wieder freigeben
    Semaphore                  Eingabe: A0 -> SignalSemaphore

Attempt       -$240   $FC2E68  Anfrage, ob SignalSemaphore frei
    Semaphore                  Eingabe: A0 -> SignalSemaphore
                               Ausgabe: D0 = -1, falls SignalSemaphore frei
                                        D0 = 0, falls SignalSemaphore belegt

Obtain        -$246   $FC2E98  Zugriffsanforderung an Liste von Semaphores
SemaphoreList                  Eingabe: A0 -> SignalSemaphore-Liste

Release       -$24C   $FC2F0E  Liste von SignalSemaphores freigeben
SemaphoreList                  Eingabe: A0 -> SignalSemaphore-Liste

FindSemaphore -$252   $FC2F34  SignalSemaphore suchen
                               Eingabe: A1 -> Namensstring
                               Ausgabe: D0 -> SignalSemaphore
                                        D0 = 0, wenn nicht gefunden
```

```
AddSemaphore   -$258   $FC2F24  Semaphore zur Semaphore-Liste hinzufügen
                                Eingabe: A1 -> SignalSemaphore

RemSemaphore   -$25E   $FC2F30  Semaphore aus Semaphore-Liste entfernen
                                Eingabe: A1 -> SignalSemaphore
```

7 Ausnahme-Zustände

Es sind zwei Arten von Ausnahmezuständen (Exceptions) zu unterscheiden: Ausnahmen des Prozessors und Ausnahmen von Tasks.

Ein Ausnahmezustand des Prozessors wird von verschiedenen Ereignissen herbeigeführt, z.B. Unterbrechungen, Adreßfehlern (Wortzugriff auf eine ungerade Adresse), undefinierte Befehlskodes oder auch bestimmte Prozessorbefehle (u.a. TRAP-Befehle). Jede Prozessorausnahme versetzt den Prozessor in den Supervisor-Modus und ruft eine Routine, deren Adresse im zugeordneten Ausnahme-Vektor im Speicherbereich $0 bis $400 steht. Einzelheiten können in jedem Prozessor-Handbuch nachgelesen werden.

Ein solcher Ausnahmezustand kann natürlich auch während der Bearbeitung einer Task auftreten; in diesem Kontext wird er grundsätzlich als Trap bezeichnet. Alle mit diesem Namen verbundenen Begriffe (tc_TrapAlloc, tc_TrapAble, tc_TrapData, tc_TrapCode im Task-Kontrollblock; Exec-Routinen AllocTrap, FreeTrap) beziehen sich auf solche Prozessor-Ausnahmen.

Die Task-spezifischen Ausnahmezustände werden demgegenüber als Exceptions bezeichnet. Hierzu gehören die Begriffe tc_SigExcept, tc_ExceptData und tc_ExceptCode im Task-Kontrollblock sowie die Routinen SetExcept und Exception.

Von den Ausnahme-Vektoren des Prozessors werden bei der Exec-Initialisierung nur die mit den Nummern 2 bis 47 belegt; die übrigen werden vom System nicht benützt. Die zugehörigen Routinen stehen im Bereich $FC081A bis $FC08A8. Sie legen jeweils die zugehörige Vektornummer auf den System-Stack und rufen entweder - wenn die Ausnahme im Supervisor-Modus entstanden ist - sofort die Alert-Routine oder sonst die Task-Traproutine, auf die der Zeiger tc_TrapCode im Task-Kontrollblock zeigt. Hier wird bei der Initialisierung des Task-Kontrollblocks ebenfalls die Adresse der Alert-Routine eingetragen, damit der Aufruf nicht ins Leere läuft; falls eine besondere Trap-Verarbeitungsroutine von der Task bereitgestellt wird, ist deren Adresse hier einzutragen. Der Zeiger tc_TrapData ist vorgesehen, um auf einen von der Trap-Routine benötigten Datenbereich zugreifen zu können.

Die Variable tc_TrapAlloc dient zur Buchführung über die Verwendung der 16 Trap-Befehle des Prozessors, die programmgesteuerte Traps auslösen. Jedes Bit ist einem dieser Trap-Befehle zugeordnet; er kann durch Aufruf der Routine AllocTrap belegt und durch Aufruf von FreeTrap freigegeben werden.

Die Task-spezifischen Ausnahmen (Exceptions) sind eine Art "privater Interrupt", der durch bestimmte Signale ausgelöst wird, die in der Variablen tc_SigExcept durch je ein gesetztes Bit dargestellt sind. Empfängt eine Task ein Signal, so wird geprüft, ob es sich um ein Exception-Signal handelt, d.h. ob das entsprechende Bit in tc_SigExcept gesetzt ist. Ist dies der Fall, so wird das Except-Bit in tc_Flags gesetzt. Wenn die Routine Dispatch beim nächsten Taskwechsel feststellt, daß dieses Bit gesetzt ist, ruft sie die Routine Exception, von der dann die Bearbeitung der Exception abgewickelt wird.

Die allgemeine Ausnahme-Routine Alert, die immer dann aufgerufen wird, wenn keine spezielle Ausnahme-Verarbeitungsroutine vorhanden ist, erzeugt die bekannte Guru-Meldung; in den meisten Fällen kann dann nur noch ein System-Reset den Rechner wieder in Gang bringen.

Es folgt eine Übersicht über die einschlägigen Routinen:

```
Routinenname  Offset  Adresse  Beschreibung
------------------------------------------------------------------------
Alert         -$06C   $FC2FD6  Ausnahme-Bearbeitung mit Guru-Meldung
                               Eingabe: A5 -> laufende Task
                                        D7 = Alert-Kode
                               Ausgabe: 'Guru Meditation Number'
                                        Alert-Kode.Zeiger auf laufende Task

AllocTrap     -$156   $FC1F8E  Trap-Befehl belegen
                               Eingabe: D0 = Trap-Nummer (0,...,15) oder
                                             -1, wenn beliebig
                               Ausgabe: D0 = zugewiesene Trap-Nummer oder
                                             -1, wenn keine Nummer frei

FreeTrap      -$15C   $FC1FB4  Trap-Befehl freigeben
                               Eingabe: D0 = Trap-Nummer (0,...,15)
```

```
SetExcept     -$138   $FC1E18  Exception-Signal definieren
                               Eingabe: D0 = neue Exception-Signale
                                        D1 = Signalmaske
                               Ausgabe: D0 = alte Exception-Signale

Exception     -$042   $FC0FCE  Task-Exception-Abwicklungsroutine
                               Eingabe: A3 -> Task

Supervisor    -$01E   $FC08AA  Routine mit Supervisor-Status aufrufen
                               (Abschluß der Routine mit RTE!)
                               Eingabe: A5 -> Routine
```

8 Bibliotheken (Libraries)

Das gesamte Betriebssystem des Amiga enthält mehr als 400 System-Routinen, die in Libraries zusammengefaßt sind. Mit der aktuellen Kickstart-Diskette werden folgende 8 Libraries in den Speicher geladen:

Name	Routinen	Adresse
exec.library	105	$FC00B6
expansion.library	25	$FC4AFC
ramlib.library	11	$FC49CC
graphics.library	109	$FC5378
layers.library	29	$FE0D90
intuition.library	78	$FD3F5C
mathffp.library	16	$FE424C
dos.library	37	$FF425A

Weitere Libraries befinden sich auf der System-Diskette im Directory 'libs' und werden bei Bedarf nachgeladen (u.a. icon.library, diskfont.library). Darüber hinaus hat der Anwender die Möglichkeit, selbst Libraries zu konstruieren (z.B. mit BASIC-Erweiterungen).

Libraries bestehen aus einem unveränderlichen und einem veränderlichen Teil. Bei den System-Libraries befindet sich der unveränderliche Teil im ROM bzw. im schreibgeschützten Kickstart-RAM, wir sprechen kurz vom Kern der Library. Bei der

Initialisierung der Library wird ein RAM-Bereich reserviert, in dem der veränderliche Teil aus Daten, die im Kern stehen, erzeugt wird. Dieser Teil besteht aus einer Struktur mit Knoten, einer Sprungliste und einem Datenbereich für Variablen. Der Knoten wird in die Library-Liste, deren List Header im Exec-Datenbereich steht (Offset $17A), eingegliedert.

Der Library-Kern enthält am Anfang eine sogenannte Resident-Struktur von folgender Gestalt:

```
DC.W    rt_Matchword    Erkennungsmarke: Kode $4AFC
DC.L    rt_Matchtag     Zeiger auf Erkennungsmarke
DC.L    rt_Endskip      Zeiger auf Ende des Kerns
DC.B    rt_Flags        Flags für Initialisierung
DC.B    rt_Version      Versionsnummer
DC.B    rt_Type         Typ (Library: 9)
DC.B    rt_Pri          Priorität
DC.L    rt_Name         Zeiger auf Namen
DC.L    rt_IDString     Zeiger auf Identifizierungs-Text
DC.L    rt_Init         Zeiger auf Initialisierungsdaten
```

Diese Resident-Struktur wird nicht nur für Libraries, sondern für alle im Speicher "residenten" Module (Gerätetreiber, Resourcen u.a.) verwendet. Das am Anfang stehende Kodewort $4AFC ist ein "illegaler Op-Kode" des Prozessors; das nachfolgende Langwort enthält die Adresse dieses Kodes. Exec durchsucht bei der Initialisierung den gesamten in Frage kommenden Adreßbereich nach derartigen Kode-Kombinationen und legt sich eine Tabelle aller residenten Moduln an. Das wichtigste Initialisierungsflag ist das AutoInit-Flag (Bit 7 von rt_Flags). Ist es nicht gesetzt, so zeigt rt_Init auf eine Routine, die die Initialisierung der Library ausführt. Ist es gesetzt, so zeigt rt_Init auf einen Initialisierungs-Datensatz im Kern; die Initialisierung wird in diesem Fall von den Exec-Routinen MakeLibrary und InitStruct mit Hilfe dieser Daten durchgeführt. Der Kern enthält außerdem eine Tabelle, in der die Adressen aller Library-Routinen aufgelistet sind. Mit Hilfe dieser Tabelle wird bei der Initialisierung die Sprungliste generiert. Der immer im RAM stehende Teil der Library hat folgende Form:

```
        jmp ...         Sprungliste
        jmp ...
         :
         :
        jmp ...
struct  lib_Node        Knoten
DC.W    lib_Flags       Flags für Prüfsumme u.a.
DC.W    lib_NegSize     Länge des Teils vor dem Knoten (Bytes)
DC.W    lib_PosSize     Länge des Teils ab dem Knoten (Bytes)
DC.W    lib_Version     Versionsnummer
DC.W    lib_Revision    Revisionsnummer
DC.L    lib_IDString    Zeiger auf Identifications-Text
DC.L    lib_Sum         Prüfsumme der Sprungliste
DC.W    lib_OpenCnt     Zähler für Library-Benutzer
         :              Beginn des Datenbereichs
         :
```

Der Knoten enthält, wie bekannt, auch einen Zeiger auf den Namen der Library, so daß die Library mittels der Routine FindName gesucht werden kann. Wenn eine Library-Routine gerufen werden soll, muß die Library geöffnet sein. Das Öffnen geschieht mittels der Exec-Routine OpenLibrary. Diese Routine sucht die Library und gibt einen Zeiger auf den Knoten zurück. Die gewünschte Funktion kann dann über den (negativen) Offset in der Sprungliste aufgerufen werden. Wird die Library nicht mehr benötigt, wird sie mittels der Routine CloseLibrary wieder geschlossen. Der Zähler lib_OpenCnt wird bei jedem Öffnen um 1 erhöht und bei jedem Schließen um 1 erniedrigt. Steht er auf Null, so ist die Library im Augenblick nicht benutzt und kann - z.B. wenn der Speicher knapp wird - aus dem Speicher entfernt werden. Der Zähler hat also eine ähnliche Funktion wie die Verschachtelungszähler für Taskwechsel- und Interruptsperren. Die ersten 4 Positionen in der Sprungliste sind reserviert für Aufrufe, die in jeder Library vorhanden sein müssen:

```
Offset -$006   Open    Library-spezifische Open-Routine,
                       wird von OpenLibrary gerufen

Offset -$00C   Close   Library-spezifische Close-Routine,
                       wird von CloseLibrary gerufen
```

```
Offset -$Ø12    Expunge   Routine zur Entfernung der Library bzw.
                          ihres im RAM liegenden Teils aus dem
                          Speicher, wird von RemLibrary gerufen

Offset -$Ø18    Extfunct  reserviert
```

Die Routinen Open und Close müssen mindestens den Open-Zähler lib_OpnCnt erhöhen bzw. erniedrigen; dies ist in der Regel das einzige, was sie tun. Expunge darf erst dann den für die Library reservierten Speicher freigeben, wenn der Open-Zähler auf Null steht.

Für den Umgang mit Libraries stellt Exec folgende Routinen zur Verfügung:

```
Routinenname  Offset  Adresse  Beschreibung
-----------------------------------------------------------------------
MakeLibrary   -$Ø54   $FC14EC  Library aus Kern erzeugen
                               Eingabe: AØ -> Tabelle der Routinenadressen
                                        A1 -> Initialisierungsdaten
                                        A2 -> Initialisierungsroutine
                                        DØ = Bytezahl f.Knoten u.Datenber.
                                        D1 -> SegList, falls benötigt
                               Ausgabe: DØ -> Zeiger auf Library-Knoten

SumLibrary    -$1AA   $FC1498  Prüfsumme für Sprungliste ermitteln
                               Eingabe: A1 -> Library-Knoten

MakeFunctions -$Ø5A   $FC1576  Sprungliste aus Vektortabelle generieren
                               (wird von MakeLibrary gerufen)
                               Eingabe: A1 -> Vektortabelle
                                        A2 = Basis für Adreßberechnung

SetFunction   -$1A4   $FC147A  Sprungadresse in Sprungliste ändern
                               Eingabe: A1 -> Library-Knoten
                                        AØ = Offset in Sprungliste
                                        DØ = neue Sprungadresse
                               Ausgabe: DØ = alte Sprungadresse

AddLibrary    -$18C   $FC14ØC  Library zur Library List hinzufügen
                               Eingabe: A1 -> Library-Knoten
```

```
RemLibrary    -$192   $FC141A  Library aus Library List entfernen
                               Eingabe: A1 -> Library-Knoten
                               Ausgabe: D0 = 0 oder Fehlerkode

OpenLibrary   -$228   $FC1438  Library für Zugriff öffnen
                               Eingabe: A1 -> Library-Name
                                        D0 = Version (Mindestwert)
                               Ausgabe: D0 -> Library-Knoten
                                        D0 = 0, wenn nicht gefunden

CloseLibrary  -$19E   $FC1466  Library für Zugriff schließen
                               Eingabe: A1 -> Library-Knoten
```

9 Gerätetreiber (Devices), Ein-/Ausgabe

Der Verkehr des Betriebssystems mit Peripheriegeräten erfolgt über die Gerätetreiber (Devices). Das sind spezielle Libraries, die alle für ein spezielles Gerät notwendigen Betriebsroutinen enthalten. Folgende Devices sind im Kickstart-Ram enthalten:

Name	Adresse
keyboard.device	$FE502E
gameport.device	$FE507A
timer.device	$FE90EC
audio.device	$FC34CC
input.device	$FE50C6
console.device	$FE510E
trackdisk.device	$FE98E4

Diese Treiber sind genauso aufgebaut wie die in Abschnitt 8 behandelten Libraries, haben aber eine eigene Liste, deren List Header im Exec-Datenbereich (Offset $15E) steht.

Neben den obligatorischen Library-Routinen Open, Close, Expunge und Extfunct müssen die Device-Sprunglisten zwei weitere Aufrufe enthalten:

```
Offset -$Ø1E    BeginIO   Datenverkehr beginnen
Offset -$Ø24    AbortIO   Datenverkehr abbrechen
```

Damit ist in den meisten Fällen die Sprungliste auch schon zu Ende; nur das timer.device und das console.device enthalten noch zwei oder drei weitere Routinen.

Für den Umgang mit Devices gibt es folgende Exec-Routinen:

```
Routinenname  Offset  Adresse  Beschreibung
-----------------------------------------------------------------------
AddDevice      -$1BØ   $FCØ654  Device zur Device List hinzufügen
                                Eingabe: A1 -> Device-Knoten

RemDevice      -$1B6   $FCØ662  Device aus Device List entfernen
                                Eingabe: A1 -> Device-Knoten
                                Ausgabe: DØ = Ø oder Fehlerkode

OpenDevice     -$1BC   $FCØ666  Device für Zugriff öffnen
                                Eingabe: AØ -> Device-Name
                                         A1 -> IORequest-Block
                                         DØ = Nummer der Einheit
                                         D1 = Flags

CloseDevice    -$1C2   $FCØ6B4  Device für Zugriff schließen
                                Eingabe: A1 -> IORequest-Block
```

Der Verkehr mit den Devices geschieht grundsätzlich in der Weise, daß zuerst ein Ein-Ausgabe-Anforderungsblock (IORequest) erzeugt wird. Das ist eine Datenstruktur vom Typ "Message":

```
struct  io_Message
DC.L    io_Device       Zeiger auf Device-Knoten
DC.L    io_Unit         Zeiger für internen Gebrauch
DC.W    io_Command      Kommando (standard oder speziell)
DC.B    io_Flags        Flags für Optionen und Status
DC.B    io_Error        Rückmeldung
```

```
DC.L    io_Actual     Zahl der übertragenen Bytes
DC.L    io_Length     Angeforderte Bytezahl
DC.L    io_Data       Zeiger auf Datenpuffer
DC.L    io_Offset     Byte-Offset (z.B. im File)
```

Es gibt folgende Standard-Kommandos:

```
1       cmd_Reset     Device initialisieren
2       cmd_Read      Standard-Lesekommando
3       cmd_Write     Standard-Schreibkommando
4       cmd_Update    Alle Puffer schreiben
5       cmd_Clear     Alle Puffer löschen
6       cmd_Stop      IO-Aktivität unterbrechen
7       cmd_Start     IO-Aktivität nach Stop fortsetzen
8       cmd_Flush     IO-Aktivität abbrechen
```

Darüberhinaus gibt es (z.B. für das trackdisk.device) noch weitere, gerätespezifische Kommandos.

Wenn der IORequest-Block initialisiert ist, wird die Ein-/Ausgabe über folgende Exec-Routinen abgewickelt:

```
Routinenname  Offset  Adresse  Beschreibung
--------------------------------------------------------------------------
SendIO        -$1CE   $FC06CA  Ein-/Ausgabe-Vorgang einleiten, nicht auf
                               Abschluß warten
                               Eingabe: A1 -> IORequest-Block

DoIO          -$1C8   $FC06DC  Ein-/Ausgabe-Vorgang einleiten, auf Abschluß
                               warten
                               Eingabe: A1 -> IORequest-Block
                               Ausgabe: D0 = 0 oder Fehlerkode

WaitIO        -$1DA   $FC06F2  Auf Ein-/Ausgabe-Abschluß warten
                               Eingabe: A1 -> IORequest-Block
                               Ausgabe: D0 = 0 oder Fehlerkode
```

```
CheckIO      -$1D4   $FC074E  Ein-/Ausgabe-Status prüfen
                              Eingabe: A1 -> IORequest-Block
                              Ausgabe: D0 = 0, wenn noch nicht abgeschlossen
                                       D0 -> IORequest-Block nach Abschluß

AbortIO      -$1E0   $FC076A  Ein-/Ausgabe abbrechen
                              Eingabe: A1 -> IORequest-Block
```

10 Spezielle Libraries für Hardware-Zugriffe (Resources)

Für unmittelbare Zugriffe auf bestimmte Hardware-Komponenten stehen eine Reihe von Routinen zur Verfügung, die in sogenannten Resources zusammengefaßt sind. Das System kennt folgende Resources:

Name	Adresse	Zweck
disk.resource	$FC4794	Zugriff auf Diskettenlaufwerk
cia.resource	$FC450C	Zugriff auf CIA-8520 Register
potgo.resource	$FE4880	Zugriff auf Potentiometer-Register
misc.resource	$FE4774	Zugriff auf seriellen und parallelen Port
keymap.resource	$FE4FE4	Zugriff auf Tastaturkode-Tabelle

Die Resources sind ebenso strukturiert wie Libraries und auch ganz analog zu verwenden. Wegen ihrer sehr speziellen Bedeutung kann auf sie an dieser Stelle nicht näher eingegangen werden.

11 Weitere Exec-Routinen

Neben den bisher beschriebenen Routinen, die von Exec selbst zur Abwicklung seiner Systemaufgaben benützt werden, sind über die Exec-Sprungliste noch weitere Routinen erreichbar, die als Hilfsroutinen für andere Libraries (graphics, intuition usw.) und für Anwenderprogramme bereitgestellt werden:

Routinenname	Offset	Adresse	Beschreibung
SetSR	-$090	$FC1122	Statusregister des Prozessors setzen Eingabe: D0 := Neuer Inhalt des SR D1 := Bitmaske für Änderung Ausgabe: D0 = Alter Inhalt des SR
GetCC	-$210	$FC1140	Statusflags des Prozessors lesen Ausgabe: D0 = Statusflags X,N,Z,V,C
SuperState	-$096	$FC1148	Supervisor-Status mit User-Stack Ausgabe: D0 = Alter Systemstackzeiger
UserState	-$09C	$FC1174	Rückkehr von SuperState Eingabe: D0 := Alter Systemstackzeiger
Debug	-$072	$FC232E	Aufruf des residenten Debuggers ROMWack
CopyMem	-$270	$FC2F44	Speicherkopierroutine Eingabe: A0 -> Quellbereich A1 -> Zielbereich D0 := Länge (Bytes)
CopyMemQuick	-$276	$FC2F40	Schnelle Variante von CopyMem unter der Bedingung, daß A0, A1 und D0 durch 4 teilbare Werte enthalten

CopyMem und CopyMemQuick sind leider nicht in voller Allgemeinheit anwendbar:

Wenn sich Quell- und Zielbereich überlappen, kann nur nach unten kopiert werden. Es muß dann also A1 < A0 sein, sonst werden noch nicht kopierte Daten des Quellbereichs durch den Kopiervorgang überschrieben.

12 Der Systemstart

Es folgt hier eine Auflistung der Vorgänge im Rechner vom Einschalten der Netzspannung bzw. vom Reset durch CTRL-A-A bis zur Übernahme der Kontrolle durch AmigaDOS.

(a) Vorgänge bis zum Start von Exec: (nur Amiga 1000)

- Leuchtdiode (LED) blinkt 5 mal als Bestätigung für den Aufruf des Boot-ROM, der Bildschirm wird dunkelgrau.
- Interrupts und DMA werden abgeschaltet, LED wird dunkel.
- Es wird durch Berechnung der Prüfsumme über den gesamten Bereich von $FC0000 bis $FFFFFF geprüft, ob die Kickstart-Systemroutinen schon geladen und in Ordnung sind; ist dies der Fall, so werden die restlichen Schritte von (a) übergangen und es wird sofort Exec gerufen (dies geschieht z.B. in aller Regel nach dem Reset mit CTRL-A-A).
- Es folgt eine eingehende Überprüfung des Kickstart-RAM-Bereichs auf Fehler in den Speicherbausteinen. Wird ein Fehler festgestellt, wird der Bildschirm blaugrün und ein Neustart eingeleitet.
- Anschließend wird der RAM-Bereich von $0 bis $3FFFF in der gleichen Weise geprüft. Ein Fehler bewirkt grünen Bildschirm und Neustart.
- Nun wird ein Test der vier Audio-Kanäle veranstaltet, indem eine kurze Tonfolge über wechselnde Kanäle ausgegeben wird.
- Die Ausnahmevektoren 2 bis 47 werden auf eine kurze Routine gerichtet, die einen gelben Bildschirm und Neustart veranlaßt, falls während der Anlauf-Phase eine Ausnahme (Interrupt, Busfehler etc.) eintritt.
- Das interne Laufwerk wird angeworfen und Sektor 0 einer eingelegten Diskette in den Speicher gelesen. Ist keine Diskette vorhanden oder beginnt der Sektor 0 nicht mit der Zeichenfolge 'KICK', so wird die Graphik mit der Hand, die eine Kickstart-Disk hält, ausgegeben. Dann wird auf einen Diskwechsel gewartet und der Schritt wiederholt.

- Wurde die Kickstart-Diskette erkannt, so wird deren Inhalt - von Spur 0 an aufsteigend - in einen Spurpuffer gelesen und von dort aus sektorweise dekodiert und in das Kickstart-RAM geschrieben. Hierbei werden die Sektorheader und die Prüfsummen auf ihre Richtigkeit geprüft. Wird ein Fehler festgestellt, so wird erneut die Graphik ausgegeben und ein Diskwechsel abgewartet, dann der vorhergehende Schritt wiederholt.
- Der Bildschirm wird schwarz, das Boot-ROM wird abgeschaltet und Exec gestartet.

(b) Initialisierung von Exec:

(00D2) Stackzeiger wird initialisiert. Es wird geprüft, ob Exec bei $F00000 beginnt; wenn nicht, ob bei $F00000 die Modul-Anfangsmarke $1111 steht. Gegebenenfalls wird dieser Modul in die Initialisierung einbezogen.

(00FE) LED wird dunkel, Interrupts und DMA gesperrt, Bildschirm dunkelgrau.

(0136) Ausnahmevektoren 2 bis 47 werden vorläufig auf den Initialisierungs Fehlerausgang gesetzt.

(014C) Es wird geprüft, ob Exec schon initialisiert und in Ordnung ist:

- SysBase in Adresse 4 geradzahlig?
- SysBase + ChkBase = 0?
- Prüfsumme von Offset $22 bis $53 = -1?

Ist eine dieser Bedingungen nicht erfüllt, so wird bei 01CE mit der Speicherinitialisierung fortgefahren.

(0172) Es wird geprüft, ob der Abfangvektor ColdCapture gesetzt ist; wenn ja, wird ab hier über diesen Vektor verzweigt. Der ColdCapture-Vektor wird ebenso wie die beiden folgenden Vektoren CoolCapture und WarmCapture von Exec nicht berührt. Die Vektoren sind vom Boot-ROM-Speichertest her zunächst mit 0 belegt; solange sie nicht anderweitig besetzt werden, werden sie beim Initialisierungslauf ignoriert. Sie können aber vom Anwender belegt werden und bewirken dann bei jedem Reset durch CTRL-A-A eine entsprechende Verzweigung.

(0184) LED wird hell. Die Exec-Datenprüfung wird fortgesetzt:

- Stimmt die Versionsnummer im Exec-Rambereich überein mit der im Kickstart-Header?

- Liegt der abgespeicherte Wert der Chip-Memory-Obergrenze MaxLocMem im Bereich von 256 kByte bis 512 kByte?
- Falls ein Fast-Memory-RAM vorhanden ist: Liegt seine Obergrenze im Bereich von $C40000 bis $DC0000? Ist die Obergrenze ein Vielfaches von 256 kByte?

Ist eine dieser Bedingungen nicht erfüllt, so wird mit dem nächsten Schritt fortgefahren; andernfalls wird die Ermittlung der Speichergrenzen und die Speicherlöschung übersprungen und bei 0240 weitergemacht.

(01CE) Der Bereich von $C00000 bis $DC0000 wird daraufhin geprüft, ob sich dort RAM befindet. Wenn ja, wird SysBase auf $C00000 gesetzt und der Exec-Datenbereich dort angelegt. Die Obergrenze wird in MaxExtMem gespeichert. Der Bereich von $0 bis $200000 (2 MB) wird auf das Ende des RAM-Bereichs untersucht; das Ergebnis wird als MaxLocMem abgespeichert. Sollte es kleiner als 256 kByte sein, so wird der Bildschirm grün und der Rechner geht in eine Endlosschleife, die nur die LED blinken läßt.

(0240) Der Bildschirm wird mittelgrau. Hier beginnt die Initialisierung des Exec-Datenbereichs:

(02A8) Prozessorbestückung ermitteln und in AttnFlags abspeichern,

(02B0) System-List-Headers initialisieren,

(036A) Exec-Library Header im RAM installieren,

(0370) Exec-Sprungliste aufbauen,

(0380) Memory Region Header und Speicher-Freiliste einrichten,

(03C6) Exec Library zur Library-Liste hinzufügen,

(03CC) Ausnahmevektoren endgültig belegen,

(041E) Interrupt-Server initialisieren,

(043A) Debuggersystem ROMWack initialisieren,

(043E) Prüfsumme von Exec-Offset $22 bis $53 berechnen und mit ChkSum auf -1 abgleichen.

(0454) Der Exec-Task-Kontrollblock wird angelegt und initialisiert. Die Task wird gestartet, der Supervisor-Status abgeschaltet und der Taskwechsel freigegeben.

(0500) Alle residenten Moduln im Speicherbereich $FC0000 bis $FFFFFF und im Speicherbereich $F00000 bis $F80000 werden gesucht und ihre Anfangsadressen in eine Tabelle eingetragen. Die Anfangsadresse der Tabelle steht in ResModules (Offset $12C) im Exec-Datenbereich. Gibt es von einem Modul mehrere Versionen, so wird nur die letzte berücksichtigt.

(050C) LED wird hell. Hier ist wieder ein Abfangvektor eingebaut: Dieser Vektor heißt CoolCapture. Die oben beim ColdCapture-Vektor gegebene Beschreibung gilt hier entsprechend.

(051E) Alle residenten Moduln werden initialisiert. Dies besorgt die Exec-Routine InitCode in Abhängigkeit von Flags und Parametern in dem jeweiligen Modul. Als letzter Modul wird die DOS-Bootroutine Strap aufgerufen. Sie prüft das interne Laufwerk auf das Vorhandensein einer Disk, deren Sektor 0 mit der Zeichenfolge 'DOS' beginnt. Ist dies nicht der Fall, so wird mit der bekannten Graphik die Workbench-Disk angefordert. Wird die DOS-Diskette erkannt, so wird die DOS-Initialisierungsroutine von Sektor 0 und 1 geladen und gestartet. Diese öffnet die im Kickstart-RAM bereits vorhandene dos.library und übergibt die Kontrolle an AmigaDOS.

Falls die Übergabe an AmigaDOS aus irgendeinem Grund fehlschlägt und InitCode die Kontrolle an Exec zurückgibt, ist nochmals ein Abfangen möglich mit dem WarmCapture-Vektor. Geschieht dies nicht, wird Debug über die Exec-Sprungliste (Offset -$072) aufgerufen. Dieser Vektor zeigt, wenn er nicht (z.B. mittels CoolCapture) anders gesetzt wurde, auf den residenten Debugger ROM-Wack, der allerdings nur mittels eines Terminals an der seriellen Schnittstelle betrieben werden kann.

13 Die Listings

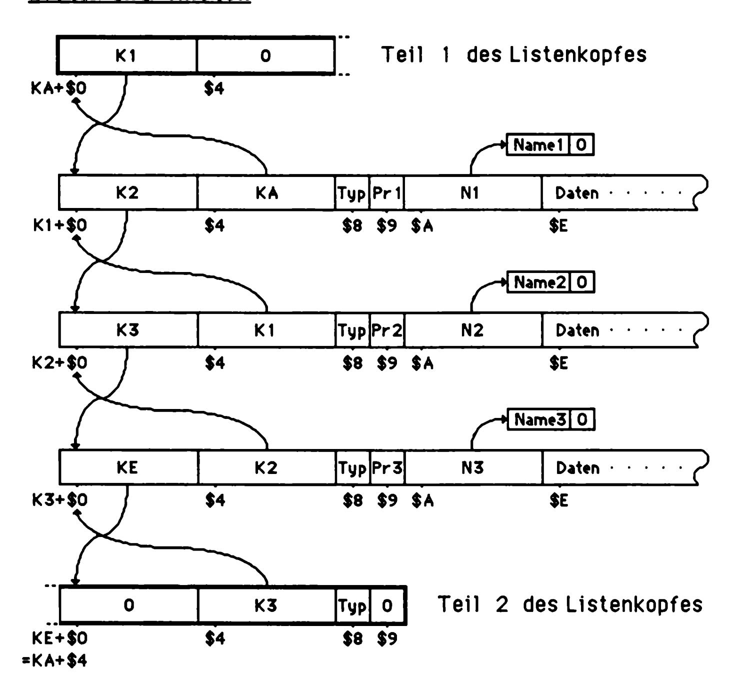
Zu den nachfolgend abgedruckten Listings ist folgendes anzumerken: Die in der ersten Spalte stehende vierstellige Hex-Zahl ist die Adresse des gelisteten Befehls ohne die beiden höchsten Stellen.

Wenn auf eine Adresse irgendwo zugegriffen wird, so steht in der zweiten Spalte die vollständige Adresse; diese Einträge sind also als Labels zu verstehen.

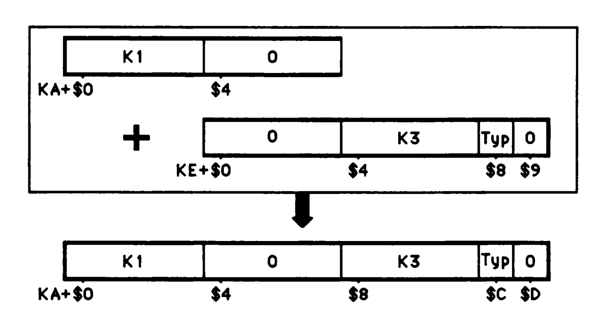
In den Befehlsoperanden sind alle Zahlenangaben im Hexagesimalsystem gemacht, deshalb ist das an sich obligatorische $-Zeichen immer weggelassen. Die Lesbarkeit wird dadurch deutlich verbessert.

In den Kommentaren sind alle Zahlenangaben ohne $-Zeichen im Dezimalsystem, solche mit $-Prefix im Hexagesimalsystem gemacht. Eine Ausnahme bilden vollständige Adressen (= Labels), die kein $-Zeichen tragen. Soweit die Original-Dokumentationen für Routinen, Variablen usw. bestimmte Namen angeben, wurden diese Namen unverändert in die Kommentare und Überschriften übernommen.

Liste und Knoten



Listenkopf (List Header)



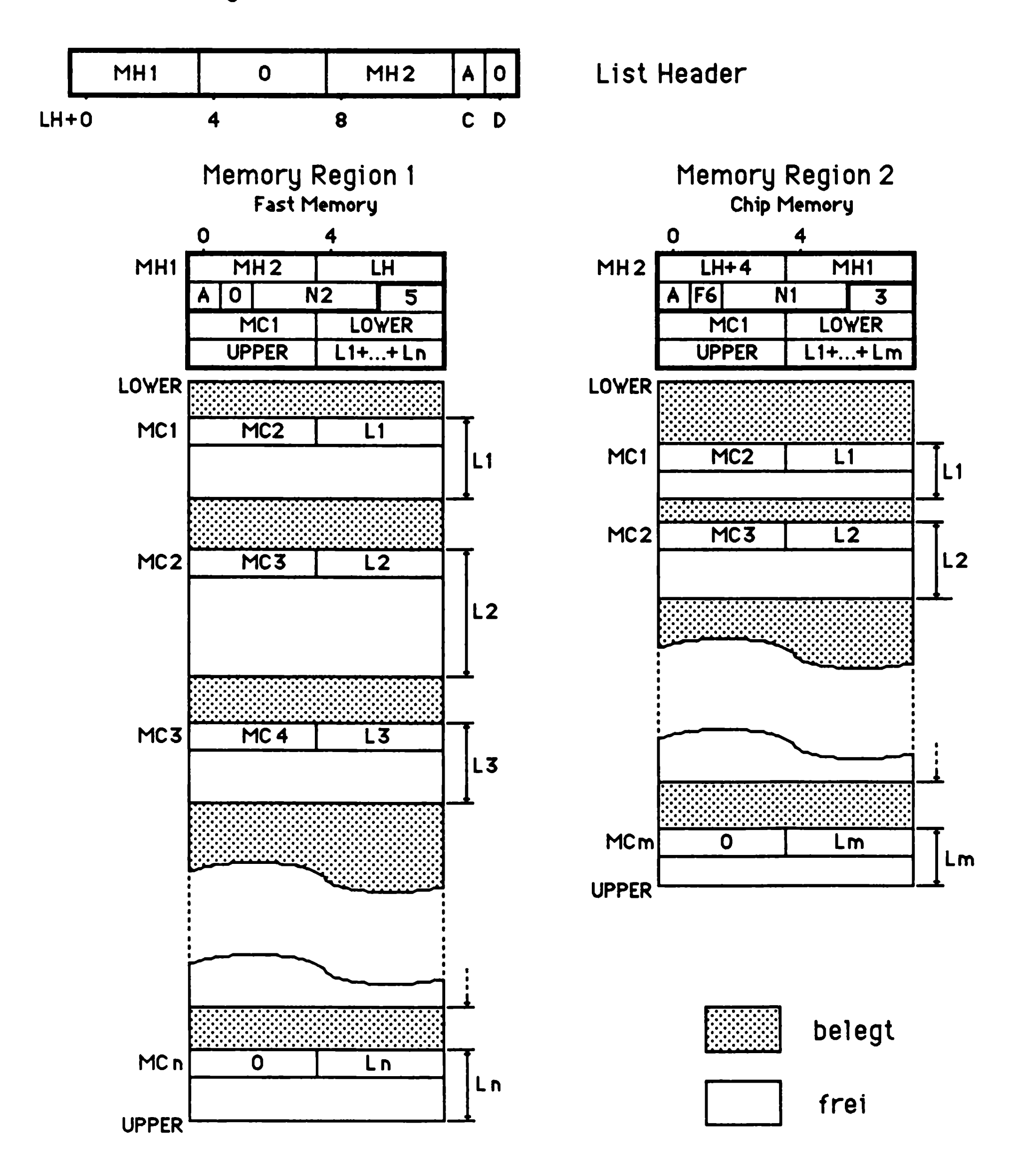
Listenkopf einer leeren Liste

KA+4 | 0 | KA | Typ | 0

KA+$0 $4 $8 $C $D

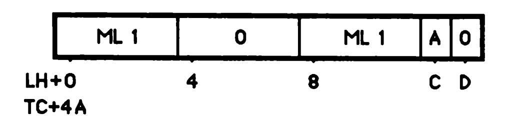
Verwaltung des freien Speichers (Free List)

(Alle Zahlenangaben in Hex)

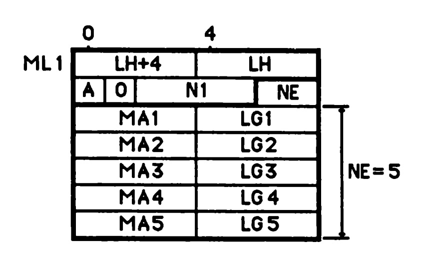


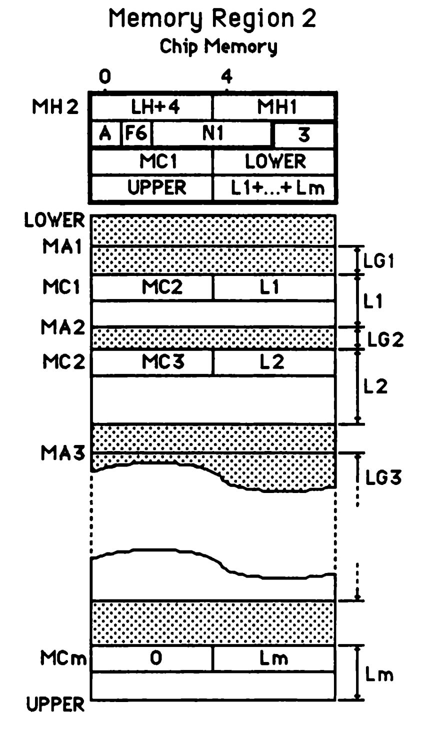
Verwaltung des belegten Speichers (Memory List)

(Alle Zahlenangaben in Hex)



List Header im Task Control Blc





Exec - Sprungliste 33 . 192

```
**********************************************************
*                                                        *
*  E X E C - S P R U N G L I S T E       3 3 . 1 9 2     *
*                                                        *
**********************************************************
```

Falls Fast Memory vorhanden, gilt Adr(F), sonst Adr(C).

Offs	Adr(C)	Adr(F)		Vektor	Name der Routine
-276	400	C00000	jmp	FC2F40	CopyMemQuick
-270	406	C00006	jmp	FC2F44	CopyMem
-26A	40C	C0000C	jmp	FC19EA	AddMemList
-264	412	C00012	jmp	FC0A3C	SumKickData
-25E	418	C00018	jmp	FC2F30	RemSemaphore
-258	41E	C0001E	jmp	FC2F24	AddSemaphore
-252	424	C00024	jmp	FC2F34	FindSemaphore
-24C	42A	C0002A	jmp	FC2F0E	ReleaseSemaphoreList
-246	430	C00030	jmp	FC2E98	ObtainSemaphoreList
-240	436	C00036	jmp	FC2E68	AttemptSemaphore
-23A	43C	C0003C	jmp	FC2E04	ReleaseSemaphore
-234	442	C00042	jmp	FC2DB4	ObtainSemaphore
-22E	448	C00048	jmp	FC2D94	InitSemaphore
-228	44E	C0004E	jmp	FC1438	OpenLibrary
-222	454	C00054	jmp	FC2D72	Vacate
-21C	45A	C0005A	jmp	FC2D5C	Procure
-216	460	C00060	jmp	FC181A	TypeOfMem
-210	466	C00066	jmp	FC1140	GetCC
-20A	46C	C0006C	jmp	FC20E8	RawDoFmt
-204	472	C00072	jmp	FC222E	RawPutChar
-1FE	478	C00078	jmp	FC2202	RawMayGetChar

```
-1F8  47E  C0007E  jmp  FC21F8  RawIOInit
-1F2  484  C00084  jmp  FC1C34  OpenResource
-1EC  48A  C0008A  jmp  FC1C30  RemResource
-1E6  490  C00090  jmp  FC1C28  AddResource
-1E0  496  C00096  jmp  FC076A  AbortIO
-1DA  49C  C0009C  jmp  FC06F2  WaitIO
-1D4  4A2  C000A2  jmp  FC074E  CheckIO
-1CE  4A8  C000A8  jmp  FC06CA  SendIO
-1C8  4AE  C000AE  jmp  FC06DC  DoIO
-1C2  4B4  C000B4  jmp  FC06B4  CloseDevice
-1BC  4BA  C000BA  jmp  FC0666  OpenDevice
-1B6  4C0  C000C0  jmp  FC0662  RemDevice
-1B0  4C6  C000C6  jmp  FC0654  AddDevice
-1AA  4CC  C000CC  jmp  FC1498  SumLibrary
-1A4  4D2  C000D2  jmp  FC147A  SetFunction
-19E  4D8  C000D8  jmp  FC1466  CloseLibrary
-198  4DE  C000DE  jmp  FC1430  OldOpenLibrary
-192  4E4  C000E4  jmp  FC141A  RemLibrary
-18C  4EA  C000EA  jmp  FC140C  AddLibrary
-186  4F0  C000F0  jmp  FC1C1E  FindPort
-180  4F6  C000F6  jmp  FC1BF6  WaitPort
-17A  4FC  C000FC  jmp  FC1BDC  ReplyMsg
-174  502  C00102  jmp  FC1BAE  GetMsg
-16E  508  C00108  jmp  FC1B34  PutMsg
-168  50E  C0010E  jmp  FC1B30  RemPort
-162  514  C00114  jmp  FC1B18  AddPort
-15C  51A  C0011A  jmp  FC1FB4  FreeTrap
-156  520  C00120  jmp  FC1F8E  AllocTrap
-150  526  C00126  jmp  FC1FFC  FreeSignal
-14A  52C  C0012C  jmp  FC1FC4  AllocSignal
-144  532  C00132  jmp  FC1E48  Signal
-13E  538  C00138  jmp  FC1ED0  Wait
-138  53E  C0013E  jmp  FC1E18  SetExcept
-132  544  C00144  jmp  FC1E22  SetSignal
-12C  54A  C0014A  jmp  FC1DC8  SetTaskPri
-126  550  C00150  jmp  FC1D74  FindTask
-120  556  C00156  jmp  FC1CF4  RemTask
-11A  55C  C0015C  jmp  FC1C48  AddTask
-114  562  C00162  jmp  FC165A  FindName
-10E  568  C00168  jmp  FC1634  Enqueue
```

```
-108  56E  C0016E  jmp  FC161E  RemTail
-102  574  C00174  jmp  FC160E  RemHead
-0FC  57A  C0017A  jmp  FC1600  Remove
-0F6  580  C00180  jmp  FC15E8  AddTail
-0F0  586  C00186  jmp  FC15D8  AddHead
-0EA  58C  C0016C  jmp  FC15AC  Insert
-0E4  592  C00192  jmp  FC19AC  FreeEntry
-0DE  598  C00198  jmp  FC191E  AllocEntry
-0D8  59E  C0019E  jmp  FC18D0  AvailMem
-0D2  5A4  C001A4  jmp  FC17F0  FreeMem
-0CC  5AA  C001AA  jmp  FC1840  AllocAbs
-0C6  5B0  C001B0  jmp  FC1794  AllocMem
-0C0  5B6  C001B6  jmp  FC1704  Deallocate
-0BA  5BC  C001BC  jmp  FC169C  Allocate
-0B4  5C2  C001C2  jmp  FC1320  Cause
-0AE  5C8  C001C8  jmp  FC1214  RemIntServer
-0A8  5CE  C001CE  jmp  FC11D4  AddIntServer
-0A2  5D4  C001D4  jmp  FC118E  SetIntVector
-09C  5DA  C001DA  jmp  FC1174  UserState
-096  5E0  C001E0  jmp  FC1148  SuperState
-090  5E6  C001E6  jmp  FC1122  SetSR
-08A  5EC  C001EC  jmp  FC1F60  Permit
-084  5F2  C001F2  jmp  FC1F5A  Forbid
-07E  5F8  C001F8  jmp  FC13FA  Enable
-078  5FE  C001FE  jmp  FC13EC  Disable
-072  604  C00204  jmp  FC232E  Debug
-06C  60A  C0020A  jmp  FC2FD6  Alert
-066  610  C00210  jmp  FC0B28  InitResident
-060  616  C00216  jmp  FC0AC0  FindResident
-05A  61C  C0021C  jmp  FC1576  MakeFunctions
-054  622  C00222  jmp  FC14EC  MakeLibrary
-04E  628  C00228  jmp  FC0BC8  InitStruct
-048  62E  C0022E  jmp  FC0AF0  InitCode
-042  634  C00234  jmp  FC0FCE  Exception
-03C  63A  C0023A  jmp  FC0F2A  Dispatch
-036  640  C00240  jmp  FC0EE0  Switch
-030  646  C00246  jmp  FC1F38  Reschedule
-02A  64C  C0024C  jmp  FC0E86  Schedule
-024  652  C00252  jmp  FC0E60  ExitIntr
-01E  658  C00258  jmp  FC08AA  Superviser
```

```
-Ø18  65E  CØØ25E  jmp  FC22EC  Extfunct
-Ø12  664  CØØ264  jmp  FC22EC  Expunge
-ØØC  66A  CØØ26A  jmp  FC22E8  Close
-ØØ6  67Ø  CØØ27Ø  jmp  FC22EØ  Open

*************** Ende der Exec-Sprungliste ***************
```

Exec - Library (RAM - BASE)

```
************************************************************************
*                                                                      *
*  E X E C - L I B R A R Y    ( R A M - B A S E )                      *
*                                                                      *
************************************************************************

Falls Fast Memory vorhanden ist, gilt Adr(F), sonst Adr(C).
Unter "Wert" eingetragene Angaben können teilweise variieren.

Ofs Adr(C) Adr(F)  Typ   Wert          Name
------------------------------------------------------------------------
000  676   C00276  DC.L  1CF6          ; ln_Succ       )
004  67A   C0027A  DC.L  07F0          ; ln_Pred       )
008  67E   C0027E  DC.B  9             ; ln_Type       )  ExecLib Node
009  67F   C0027F  DC.B  0             ; ln_Pri        )
00A  680   C00280  DC.L  FC00A8        ; ln_Name       )

00E  684   C00284  DC.W  0400          ; lib_Flags
010  686   C00286  DC.W  0276          ; lib_NegSize
012  688   C00288  DC.W  024C          ; lib_PosSize
014  68A   C0028A  DC.W  0021          ; lib_Version
016  68C   C0028C  DC.W  00C0          ; lib_Revision
018  68E   C0028E  DC.L  FC0018        ; lib_idString
01C  692   C00292  DC.L  66C50000      ; lib_Sum
020  696   C00296  DC.W  2             ; lib_OpenCnt

022  698   C00298  DC.W  0             ; SoftVer
024  69A   C0029A  DC.W  0             ; LowMemChkSum
026  69C   C0029C  DC.L  FFFFF989      ; ChkBase
```

```
02A  6A0  C002A0  DC.L  0                ; ColdCapture
02E  6A4  C002A4  DC.L  0                ; CoolCapture
032  6A8  C002A8  DC.L  0                ; WarmCapture

 36  6AC  C002AC  DC.L  80000            ; SysStkUpper
 3A  6B0  C002B0  DC.L  7E800            ; SysStkLower
 3E  6B4  C002B4  DC.L  80000            ; MaxLocMem

 42  6B8  C002B8  DC.L  FC2342           ; DebugEntry
 46  6BC  C002BC  DC.L  0                ; DebugData
 4A  6C0  C002C0  DC.L  0                ; AlertData

 4E  6C4  C002C4  DC.L  0                ; MaxExtMem

 52  6C8  C002C8  DC.W  FA22             ; ChkSum

 ; Interrupt-Vektoren:

 54  6CA  C002CA  DC.L  0       ; IVTBE     - Ser.Port T-Buffer leer  iv_Data
 58  6CE  C002CE  DC.L  0       ;                                     iv_Code
 5C  6D2  C006D2  DC.L  0       ;                                     iv_Node

 60  6D6  C002D6  DC.L  1EDE    ; IVDSKBLK  - Disk Block fertig
 64  6DA  C002DA  DC.L  FC4A80
 68  6DE  C002DE  DC.L  1F2C

 6C  6E2  C002E2  DC.L  0       ; IVSOFTINT - Software Interrupt
 70  6E6  C002E6  DC.L  FC1380
 74  6EA  C002EA  DC.L  0

 78  6EE  C002EE  DC.L  08E8    ; IVPORTS   - IO-Ports & Timers
 7C  6F2  C002F2  DC.L  FC12FC
 80  6F6  C002F6  DC.L  0

 84  6FA  C002FA  DC.L  0914    ; IVCOPER   - Copper
 88  6FE  C002FE  DC.L  FC12FC
 8C  702  C00302  DC.L  0
```

```
90  706  C00306  DC.L  08FE    ; IVVERTB   - Start Vertical Blank
94  70A  C0030A  DC.L  FC12FC
98  70E  C0030E  DC.L  0

9C  712  C00312  DC.L  21FE    ; IVBLIT    - Blitter fertig
A0  716  C00316  DC.L  FC6D8E
A4  71A  C0031A  DC.L  2274

A8  71E  C0031E  DC.L  30A6    ; IVAUD0    - Audio Kanal 0 fertig
AC  722  C00322  DC.L  FC35A4
B0  726  C00326  DC.L  30D8

B4  72A  C0032A  DC.L  3112    ; IVAUD1    - Audio Kanal 1 fertig
B8  72E  C0032E  DC.L  FC35A4
BC  732  C00332  DC.L  3144

C0  736  C00336  DC.L  317E    ; IVAUD2    - Audio Kanal 2 fertig
C4  73A  C0033A  DC.L  FC35A4
C8  73E  C0033E  DC.L  31B0

CC  742  C00342  DC.L  31EA    ; IVAUD3    - Audio Kanal 3 fertig
D0  746  C00346  DC.L  FC35A4
D4  74A  C0034A  DC.L  321C

D8  74E  C0034E  DC.L  0       ; IVRBF     - Ser.Port R-Buffer voll
DC  752  C00352  DC.L  0
E0  756  C00356  DC.L  0

E4  75A  C0035A  DC.L  1EDE    ; IVDISKSYNC- Disk Sync Reg = Data
E8  75E  C0035E  DC.L  FC4A98
EC  762  C00362  DC.L  1F42

F0  766  C00366  DC.L  092A    ; IVEXTER   - Externer Interrupt
F4  76A  C0036A  DC.L  FC12FC
F8  76E  C0036E  DC.L  0

FC  772  C00372  DC.L  0       ; IVINTEN   - Master Interrupt
100 776  C00376  DC.L  0
104 77A  C0037A  DC.L  0
```

```
108  77E  C0037E  DC.L  0940    ; IVNMI        - Nicht maskierb. Interrupt
10C  782  C00382  DC.L  FC12FC
110  786  C00386  DC.L  0

; Dynamische Systemvariable

114  78A  C0038A  DC.L  203D8   ; ThisTask       - Zeiger auf laufende Ta
118  78E  C0038E  DC.L  6A24    ; IdleCount      - Wartezyklen-Zähler
11C  792  C00392  DC.L  5B77    ; DispCount      - Dispatch-Zähler
120  796  C00396  DC.W  0010    ; Quantum        - Zeitscheibe
122  798  C00398  DC.W  000F    ; Elapsed        - abgelaufene Zeit
124  79A  C0039A  DC.W  0       ; SysFlags       - verschiedene System-Flags
126  79C  C0039C  DC.B  FF      ; IDNestCnt      - Int-Disable Ebene
127  79D  C0039D  DC.B  FF      ; TDNestCnt      - Task-Disable Ebene
128  79E  C0039E  DC.W  0       ; AttnFlags      - Flags f. Prozessortyp
12A  7A0  C003A0  DC.W  0       ; AttnResched    - Flags f. Reschedule
12C  7A2  C003A2  DC.L  1C00    ; ResModules     - Zeiger auf Modul-Liste
130  7A6  C003A6  DC.L  FC2FB4  ; TaskTrapCode   - Default Task-Trap-Rout.
134  7AA  C003AA  DC.L  FC2FB4  ; TaskExceptCode- Def. Task-Exception-Rout.
138  7AE  C003AE  DC.L  FC1CEC  ; TaskExitCode   - Def. Task-Abschluss-Rout.
13C  7B2  C003B2  DC.L  FFFF    ; TaskSigAlloc   - Vorbelegung Signalmask
140  7B6  C003B6  DC.W  8000    ; TaskTrapAlloc  - Vorbelegung Trapmaske

; System List-Headers:

142  7B8  C003B8  DC.L  08C2    ; MemList
146  7BC  C003BC  DC.L  0
14A  7C0  C003C0  DC.L  08C2
14E  7C4  C003C4  DC.W  0A00

150  7C6  C003C6  DC.L  19EA    ; ResourceList
154  7CA  C003CA  DC.L  0
158  7CE  C003CE  DC.L  1BA4
15C  7D2  C003D2  DC.W  0800

15E  7D4  C003D4  DC.L  2C1C    ; DeviceList
162  7D8  C003D8  DC.L  0
166  7DC  C003DC  DC.L  4D6AC
16A  7E0  C003E0  DC.W  0300
```

```
16C  7E2  C003E2  DC.L  07E6    ; IntrList
170  7E6  C003E6  DC.L  0
174  7EA  C003EA  DC.L  07E2
178  7EE  C003EE  DC.W  0200

17A  7F0  C003F0  DC.L  0676    ; LibList
17E  7F4  C003F4  DC.L  0
182  7F8  C003F8  DC.L  01CE6C
186  7FC  C003FC  DC.W  0900

188  7FE  C003FE  DC.L  481C8   ; PortList
18C  802  C00402  DC.L  0
190  806  C00406  DC.L  1D5A0
194  80A  C0040A  DC.W  0400

196  80C  C0040C  DC.L  0810    ; TaskReady
19A  810  C00410  DC.L  0
19E  814  C00414  DC.L  080C
1A2  818  C00418  DC.W  0100

1A4  81A  C0041A  DC.L  A708    ; TaskWait
1A8  81E  C0041E  DC.L  0
1AC  822  C00422  DC.L  1E07A
1B0  826  C00426  DC.W  0100

1B2  828  C00428  DC.L  082C    ; SoftInt (Pri -32)
1B6  82C  C0042C  DC.L  0
1BA  830  C00430  DC.L  0828
1BE  834  C00434  DC.W  0B00
1C0  836  C00436  DC.W  0

1C2  838  C00438  DC.L  083C    ; SoftInt (Pri -16)
1C6  83C  C0043C  DC.L  0
1CA  840  C00440  DC.L  0838
1CE  844  C00444  DC.W  0B00
1D0  846  C00446  DC.W  0
```

```
1D2  848  C00448  DC.L  084C   ; SoftInt (Pri 0)
1D6  84C  C0044C  DC.L  0
1DA  850  C00450  DC.L  0848
1DE  854  C00454  DC.W  0B00
1E0  856  C00456  DC.W  0

1E2  858  C00458  DC.L  085C   ; SoftInt (Pri 16)
1E6  85C  C0045C  DC.L  0
1EA  860  C00460  DC.L  0858
1EE  864  C00464  DC.W  0B00
1F0  866  C00466  DC.W  0

1F2  868  C00468  DC.L  086C   ; SoftInt (Pri 32)
1F6  86C  C0046C  DC.L  0
1FA  870  C00470  DC.L  0868
1FE  874  C00474  DC.W  0B00
200  876  C00476  DC.W  0

202  878  C00478  DC.L  -1     ; LastAlert
206  87C  C0047C  DC.L  0
20A  880  C00480  DC.L  0
20E  884  C00484  DC.L  0

212  888  C00488  DC.B  32     ; VBlankFrequency
213  889  C00489  DC.B  32     ; PowerSupplyFrequency

214  88A  C0048A  DC.L  088E   ; SemaphoreList
218  88E  C0048E  DC.L  0
21C  892  C00492  DC.L  088A
220  896  C00496  DC.W  0F00

222  898  C00498  DC.L  0      ; KickMemPtr
226  89C  C0049C  DC.L  0      ; KickTagPtr
22A  8A0  C004A0  DC.L  0      ; KickChkSum
```

```
22E  8A4   C004A4  DC.B   0,0,0,0,0,0,0,0,0,0    ; ExecBaseReserved

238  8AE   C004AE  DC.B   0,0,0,0,0,0,0,0,0,0    ; ExecBaseNewReserved
                   DC.B   0,0,0,0,0,0,0,0,0,0

******************* Ende der Exec-Library RAM-Base ********************

; Memory Region Header (Chip Memory)

24C  8C2  DC.L   07BC             ; ln_Succ
250  8C6  DC.L   07B8             ; ln_Pred
254  8CA  DC.B   0A               ; ln_Type
255  8CB  DC.B   F6               ; ln_Pri
256  8CC  DC.L   FC0326           ; ln_Name
25A  8D0  DC.W   3                ; mh_Attributes
25C  8D2  DC.L   1CD80            ; mh_First
260  8D6  DC.L   08E8             ; mh_Lower
264  8DA  DC.L   07E800           ; mh_Upper
268  8DE  DC.L   02A160           ; mh_Free

26C  8E2  DC.W   0
26E  8E4  DC.L   0

*************************************************************************
```

Tabelle der residenten Moduln

```
************************************************************
*                                                          *
* Tabelle der residenten Moduln                            *
*                                                          *
************************************************************

1C00  DC.L   FC00B6           exec.library
1C04  DC.L   FC4AFC           expansion.library
1C08  DC.L   FE4880           potgo.resource
1C0C  DC.L   FE4FE4           keymap.resource
1C10  DC.L   FC450C           cia.resource
1C14  DC.L   FC4794           disk.resource
1C18  DC.L   FE4774           misc.resource
1C1C  DC.L   FE49CC           ramlib.library
1C20  DC.L   FC5378           graphics.library
1C24  DC.L   FE502E           keyboard.device
1C28  DC.L   FE507A           gameport.device
1C2C  DC.L   FE90EC           timer.device
1C30  DC.L   FC34CC           audio.device
1C34  DC.L   FE50C6           input.device
1C38  DC.L   FE0D90           layers.library
1C3C  DC.L   FE510E           console.device
1C40  DC.L   FE98E4           trackdisk.device
1C44  DC.L   FD3F5C           intuition.library
1C48  DC.L   FC323A           alert.hook
1C4C  DC.L   FE424C           mathffp.library
1C50  DC.L   FEB400           workbench.task
1C54  DC.L   FF425A           dos.library
1C58  DC.L   FE8884           strap
1C5C  DC.L   0

************************************************************
```

Abkürzungen

AG	General Purpose Alert Codes
AN	Specific Dead-End Alerts
AO	Alert Objects
AT	Alert Types
CMD	IO-Command
DD	Device Data Structure
IO	IO-Request Structure
IOERR	IO-Error
IS	Interrupt Structure
IV	Exec Internal Interrupt Vector
LH	List Header Structure
LIB	Standard Library Data Structure
LN	List Node Structure
MC	Memory Chunk Structure
ME	Memory Entry Structure
MH	Memory Region Header Structure
ML	Memory List Structure
MLH	Minimal List Header Structure
MLN	Minimal List Node Structure
MN	Message Structure
MP	Message Port Structure
NT	Node Type
RT	Resident Module Tag
SH	Software Interrupt List Header
SM	Semaphore Structure
SS	Signal Semaphore Structure
SSR	Signal Semaphore Request Structure
TC	Task Control Structure
TS	Task State
UNIT	IO-Unit Structure

Amiga Exec 33 . 192

```
0000 ;*********************************************************************
0000 ;*                                                                   *
0000 ;*          A M I G A   E X E C   3 3 . 1 9 2                        *
0000 ;*                                                                   *
0000 ;*********************************************************************
0000
0000 FC0000  DC.W    1111                 ; Modul-Anfangsmarke
0002
0002         jmp     FC00D2               ; ---> Initialisierung
0008
0008         DC.W    0
000A         DC.W    -1
000C         DC.W    21
000E         DC.W    B4
0010
0010 FC0010  DC.W    21                   ; Version
0012         DC.W    C0                   ; Revision
0014         DC.L    -1
0018
0018 FC0018  DC.B    'exec 33.192 (8 Oct 1986)',0D,0A,0,0 ; idString
0034
0034         DC.L    -1
0038
0038         DC.B    0D,0A,0A,'AMIGA ROM Operating System and Libraries',0D,0A
0065         DC.B    'Copyright (C) 1985, Commodore-Amiga, Inc.',0D,0A
0090         DC.B    'All Rights Reserved.',0D,0A,0,0
00A8
00A8 FC00A8  DC.B    'exec.library',0,0 ; libName
00B6
00B6 ;------ Library Header Structure
00B6
```

```
00B6 FC00B6 DC.W     4AFC              ; rt_Matchword
00B8        DC.L     FC00B6            ; rt_Matchtag
00BC        DC.L     FC323A            ; rt_Endskip
00C0        DC.B     0                 ; rt_Flags
00C1        DC.B     21                ; rt_Version
00C2        DC.B     9                 ; rt_Type
00C3        DC.B     78                ; rt_Pri
00C4        DC.L     FC00A8            ; rt_Name
00C8        DC.L     FC0018            ; rt_idString
00CC        DC.L     FC00D2            ; rt_Init
00D0
00D0        reset
00D2
00D2 ;------ Initialisierung
00D2
00D2 FC00D2 lea      40000,a7          ; Stackzeiger auf 256k setzen
00D8        move.l   #20000,d0         ; Zähler := 131072
00DE FC00DE subq.l   #1,d0             ; Warteschleife ca. 0.5 s
00E0        bgt.s    FC00DE
00E2        lea      -E4(pc),a0        ; a0 := Modul-Anfang
00E6        lea      F00000,a1         ; a1 := F00000
00EC        cmpa.l   a1,a0             ; Modul-Anfang = F00000?
00EE        beq.s    FC00FE            ; ja: ->
00F0        lea      C(pc),a5          ; a5 := ..00FE (Rückkehradresse)
00F4        cmpi.w   #1111,(a1)        ; Modul-Anfangsmarke vorhanden?
00F8        bne.s    FC00FE            ; nein: ->
00FA        jmp      2(a1)             ; ---> Modul initialisieren
00FE
00FE FC00FE move.b   #3,BFE201         ; 8520-A DDRA: Bit 0, 1 = Output
0106        move.b   #2,BFE001         ; 8520-A PRA : LED dunkel
010E        lea      DFF000,a4         ; a4 := Spezialchip Basisadresse
0114        move.w   #7FFF,d0
0118        move.w   d0,9A(a4)         ; INTENA: Alle IR-Enable-Bits := 0
011C        move.w   d0,9C(a4)         ; INTREQ: Alle IR-Req-Bits := 0
0120        move.w   d0,96(a4)         ; DMACON: Alle DMA-Enable-Bits := 0
0124        move.w   #200,100(a4)      ; BPLCON0: Comp.Video Color Enable
012A        move.w   #0,110(a4)        ; BPL1DAT: Bildschirmausgabe starten
0130        move.w   #444,180(a4)      ; COLOR0: Bildschirm dunkelgrau
0136
0136        movea.w  #8,a0             ; a0 := Anfang der Vektortabelle
```

```
013A          move.w   #2D,d1          ; d1 := 45 (Zähler)
013E          lea      474(pc),a1      ; a1 := FC05B4 (Fehlerausgang)
0142 FC0142   move.l   a1,(a0)+        ; Vektoren 2 bis 47 vorläufig
0144          dbra     d1,FC0142       ; auf Fehlerausgang richten
0148          bra      FC30C4          ; ---> Umleitung für Alert-Reset
014C
014C FC014C   move.l   4,d0            ; d0 von SysBase laden
0150          btst.l   #0,d0           ; Bit 0 = 1?
0154          bne.s    FC01CE          ; ja: Kaltstart ->
0156          movea.l  d0,a6           ; a6 := SysBase
0158          add.l    26(a6),d0       ; Kontrollkode ChkBase addieren
015C          not.l    d0              ; Ergebnis invertieren
015E          bne.s    FC01CE          ; nicht Null: Kaltstart ->
0160          moveq    #0,d1           ; d1 := 0 (Summenregister)
0162          lea      22(a6),a0       ; a0 := Anfang Variablenbereich
0166          moveq    #18,d0          ; d0 := 24 (Zähler)
0168 FC0168   add.w    (a0)+,d1        ; Prüfsumme über 25 Worte
016A          dbra     d0,FC0168       ; in d1 berechnen
016E          not.w    d1              ; und invertieren
0170          bne.s    FC01CE          ; Ergebnis nicht Null: Kaltstart ->
0172          move.l   2A(a6),d0       ; d0 := ColdCapture-Vektor
0176          beq.s    FC0184          ; = 0: ->
0178          movea.l  d0,a0           ; a0 := ColdCapture-Vektor
017A          lea      8(pc),a5        ; a5 := FC0184 (Rückkehradresse)
017E          clr.l    2A(a6)          ; ColdCapture-Vektor := 0
0182          jmp      (a0)            ; ---> ColdCapture-Routine
0184
0184 FC0184   bchg.b   #1,BFE001       ; 8520-A PRA: LED umschalten
018C          move.l   -17E(pc),d0     ; d0 := Version
0190          cmp.l    14(a6),d0       ; = Version im ExecLib-Header?
0194          bne.s    FC01CE          ; nein: Kaltstart ->
0196          movea.l  3E(a6),a3       ; a3 := MaxLocMem = Speichergrenze
019A          cmpa.l   #80000,a3       ; Speichergrenze > 512 k?
01A0          bhi.s    FC01CE          ; ja: Kaltstart ->
01A2          cmpa.l   #40000,a3       ; Speichergrenze < 256 k?
01A8          bcs.s    FC01CE          ; ja: Kaltstart ->
01AA          movea.l  4E(a6),a4       ; a4 := MaxExtMem
01AE          move.l   a4,d0           ; d0 := a4
01B0          beq      FC0240          ; kein FastMem vorhanden: ->
01B4          cmpa.l   #DC0000,a4      ; a4 > 13.75 MB?
```

```
Ø1BA        bhi.s   FCØ1CE          ; ja: ->
Ø1BC        cmpa.l  #C4ØØØØ,a4      ; a4 < 12.25 MB?
Ø1C2        bcs.s   FCØ1CE          ; ja: ->
Ø1C4        move.l  a4,dØ           ; dØ := a4
Ø1C6        andi.l  #3FFFF,dØ       ; Bits 18,...,31 löschen
Ø1CC        beq.s   FCØ24Ø          ; Ergebnis = Ø: ->
Ø1CE
Ø1CE ;---------------------------------   Kaltstart
Ø1CE
Ø1CE FCØ1CE lea     4ØØ,a6          ; a6 := erste freie RAM-Adresse
Ø1D2        suba.w  #-276,a6        ; Länge der Exec-Sprungliste addieren
Ø1D6        lea     CØØØØØ,aØ       ; aØ := Suchbereich Anfang
Ø1DC        lea     DCØØØØ,a1       ; a1 := Suchbereich Ende
Ø1E2        lea     6(pc),a5        ; a5 := FCØ1EA (Rückkehradresse)
Ø1E6        bra     FCØ61A          ; ---> RAM ab CØØØØØ suchen
Ø1EA
Ø1EA FCØ1EA move.l  a4,dØ           ; dØ := a4 = Ergebnis
Ø1EC        beq.s   FCØ2Ø8          ; Kein RAM vorhanden: ->
Ø1EE        movea.l #CØØØØØ,a6      ; a6 := Anfang des RAM-Bereichs
Ø1F4        suba.w  #-276,a6        ; a6 -> SysBase
Ø1F8        move.l  a4,dØ           ; dØ := Ende des Löschbereichs
Ø1FA        lea     CØØØØØ,aØ       ; aØ := Anfang des Löschbereichs
Ø2ØØ        lea     6(pc),a5        ; a5 := FCØ2Ø8 (Rückkehradresse)
Ø2Ø4        bra     FCØ6Ø2          ; ---> Speicher löschen
Ø2Ø8
Ø2Ø8 FCØ2Ø8 lea     Ø,aØ            ; aØ := Ø (Speicher-Untergrenze)
Ø2ØC        lea     2ØØØØØ,a1       ; a1 := 2 MB (Max. Speichergrenze)
Ø212        lea     6(pc),a5        ; a5 := FCØ21A (Rückkehradresse)
Ø216        bra     FCØ592          ; ---> Speichergrenze ermitteln
Ø21A
Ø21A FCØ21A cmpa.l  #4ØØØØ,a3       ; Speichergrenze < 256 kB?
Ø22Ø        bcs.s   FCØ238          ; ja: Fehler ->
Ø222        move.l  #Ø,Ø           ; 'HELP'-Flag löschen
Ø22A        move.l  a3,dØ           ; dØ := Speichergrenze
Ø22C        lea     CØ,aØ           ; aØ := Stack-Aussparung (Langworte)
Ø23Ø        lea     E(pc),a5        ; a5 := FCØ24Ø (Rückkehradresse)
Ø234        bra     FCØ6Ø2          ; ---> Speicher löschen
Ø238
Ø238 FCØ238 move.w  #CØ,dØ          ; für Bildschirmfarbe Grün
Ø23C        bra     FCØ5B8          ; ---> Fehlerausgang
```

```
0240
0240 FC0240  lea      DFF000,a0          ; a0 := Spezialchip-Basisadresse
0246         move.w   #7FFF,96(a0)       ; DMACON: Alle DMA-Enable-Bits := 0
024C         move.w   #200,100(a0)       ; BPLCON0: Comp.Video Color Enable
0252         move.w   #0,110(a0)         ; BPL1DAT: Bildschirmausgabe starten
0258         move.w   #888,180(a0)       ; COLOR0: Bildschirm mittelgrau
025E         lea      54(a6),a0          ; a0 := Anfang der Exec-Variablen
0262         movem.l  222(a6),d2-d4      ; KickMemPtr,KickTagPtr,KickChkSum retten
0268         moveq    #0,d0              ; d0 := Löschkode
026A         move.w   #7D,d1             ; d1 := Bereichslänge (LW)
026E FC026E  move.l   d0,(a0)+           ; Exec-Variablenbereich löschen
0270         dbra     d1,FC026E          ; von 54(a6) bis 24B(a6)
0274         movem.l  d2-d4,222(a6)      ; KickMemPtr usw. wiederherstellen
027A         move.l   a6,4               ; SysBase in 4 abspeichern
027E         move.l   a6,d0              ; d0 := SysBase
0280         not.l    d0                 ; invertieren
0282         move.l   d0,26(a6)          ; und als ChkBase abspeichern
0286         move.l   a4,d0              ; d0 := Ende Ext. Memory
0288         bne.s    FC028C             ; Fast Memory vorhanden: ->
028A         move.l   a3,d0              ; d0 := Ende Chip Memory
028C FC028C  movea.l  d0,a7              ; System-Stackzeiger initialisieren
028E         move.l   d0,36(a6)          ; SysStkUpper
0292         subi.l   #1800,d0           ; d0 := Stacklänge 6144 Bytes
0298         move.l   d0,3A(a6)          ; SysStkLower
029C         move.l   a3,3E(a6)          ; MaxLocMem (Chip Memory)
02A0         move.l   a4,4E(a6)          ; MaxExtMem
02A4         bsr      FC30E4             ; ---> LastAlert-Werte speichern
02A8         bsr      FC0546             ; ---> Prozessortyp ermitteln
02AC         or.w     d0,128(a6)         ; AttnFlags entsprechend setzen
02B0
02B0 ;------ System List Header installieren
02B0
02B0         lea      20(pc),a1          ; a1 := Anfang der LH-Tabelle
02B4 FC02B4  move.w   (a1)+,d0           ; d0 := Tabellenwert
02B6         beq      FC033E             ; Tabellenende erreicht: ->
02BA         lea      0(a6,d0.w),a0      ; a0 := LH-Adresse
02BE         move.l   a0,(a0)            ; lh_Head
02C0         addq.l   #4,(a0)            ; zeigt auf lh_Tail
02C2         clr.l    4(a0)              ; lh_Tail ist immer = 0
02C6         move.l   a0,8(a0)           ; lh_TailPred zeigt auf lh_Head
```

```
Ø2CA         move.w  (a1)+,dØ          ; dØ := lh_Type aus Tabelle
Ø2CC         move.b  dØ,C(aØ)          ; in Header eintragen
Ø2DØ         bra.s   FCØ2B4            ; ---> Loop
Ø2D2
Ø2D2 FCØ2D2  DC.W    142               ; MemList
Ø2D4         DC.W    A
Ø2D6         DC.W    15Ø               ; ResourceList
Ø2D8         DC.W    8
Ø2DA         DC.W    15E               ; DeviceList
Ø2DC         DC.W    3
Ø2DE         DC.W    17A               ; LibList
Ø2EØ         DC.W    9
Ø2E2         DC.W    188               ; PortList
Ø2E4         DC.W    4
Ø2E6         DC.W    196               ; TaskReady
Ø2E8         DC.W    1
Ø2EA         DC.W    1A4               ; TaskWait
Ø2EC         DC.W    1
Ø2EE         DC.W    16C               ; IntrList
Ø2FØ         DC.W    2
Ø2F2         DC.W    1B2               ; SoftInt
Ø2F4         DC.W    B
Ø2F6         DC.W    1C2               ; SoftInt
Ø2F8         DC.W    B
Ø2FA         DC.W    1D2               ; SoftInt
Ø2FC         DC.W    B
Ø2FE         DC.W    1E2               ; SoftInt
Ø3ØØ         DC.W    B
Ø3Ø2         DC.W    1F2               ; SoftInt
Ø3Ø4         DC.W    B
Ø3Ø6         DC.W    214               ; SemaphoreList
Ø3Ø8         DC.W    F
Ø3ØA         DC.W    Ø
Ø3ØC
Ø3ØC ;------ Exec Library Header
Ø3ØC
Ø3ØC FCØ3ØC  DC.B    9                 ; ln_Type
Ø3ØD         DC.B    Ø                 ; ln_Pri
Ø3ØE         DC.L    FCØØA8            ; ln_Name
Ø312         DC.W    Ø6ØØ              ; rt_Flags
```

```
0314         DC.W    0                   ; rt_NegSize
0316         DC.W    024C                ; rt_PosSize
0318         DC.W    21                  ; rt_Version
031A         DC.W    BD                  ; rt_Revision
031C         DC.L    FC0018              ; rt_idString
0320         DC.L    0                   ; rt_Sum
0324         DC.W    1                   ; rt_OpenCnt
0324
0326 FC0326  DC.B    'Chip Memory',0
0332 FC0332  DC.B    'Fast Memory',0
033E
033E FC033E  lea     2C74(pc),a0         ; a0 := FC2FB4
0342         move.l  a0,130(a6)          ; TaskTrapCode
0346         move.l  a0,134(a6)          ; TaskExceptCode
034A         move.l  #FC1CEC,138(a6)     ; TaskExitCode
0352         move.l  #FFFF,13C(a6)       ; TaskSigAlloc
035A         move.w  #8000,140(a6)       ; TaskTrapAlloc
0360         lea     8(a6),a1            ; a1 -> ln_Type (ExecLib)
0364         lea     -5A(pc),a0          ; a0 := FC030C
0368         moveq   #C,d0               ; d0 := 12 (Zähler)
036A FC036A  move.w  (a0)+,(a1)+         ; Exec Library-Header
036C         dbra    d0,FC036A           ; ins RAM kopieren
0370         movea.l a6,a0               ; a0 := SysBase
0372         lea     16CC(pc),a1         ; a1 := FC1A40 = Anfang der
0376         movea.l a1,a2               ; Library Offset Tabelle
0378         bsr     FC1576              ; ---> MakeFunctions
037C         move.w  d0,10(a6)           ; lib_NegSize eintragen
0380         move.l  a4,d0               ; Fast Memory vorhanden?
0382         beq.s   FC03A8              ; nein: ->
0384         lea     24C(a6),a0          ; a0 -> Ende Exec-Datenbereich + 1
0388         lea     -58(pc),a1          ; a1 := FC0332 -> 'Fast Memory'
038C         moveq   #0,d2               ; d2 := Priorität
038E         move.w  #5,d1               ; d1 := Attribute: Fast, Public
0392         move.l  a4,d0               ; d0 := Fast Memory Obergrenze
0394         sub.l   a0,d0               ; - Ende Exec-Datenbereich
0396         subi.l  #1800,d0            ; - System-Stack-Länge
039C         bsr     FC19EA              ; ---> AddMemList
03A0         lea     400,a0              ; a0 := Anfang des freien Chip Memory
03A4         moveq   #0,d0               ; d0 := 0
03A6         bra.s   FC03B2              ; --->
```

```
ØЗA8
Ø3A8 FCØ3A8  lea      24C(a6),aØ          ; aØ -> Ende Exec-Datenbereich + 1
Ø3AC         move.l   #-18ØØ,dØ           ; dØ := - Systemstack-Länge
Ø3B2 FCØ3B2  move.w   #3,d1               ; d1 := Attribute: Chip, Public
Ø3B6         movea.l  aØ,a2
Ø3B8         lea      -94(pc),a1          ; a1 := FCØ326 -> 'Chip Memory'
Ø3BC         moveq    #F6,d2              ; d2 := -1Ø = Priorität
Ø3BE         add.l    a3,dØ               ; dØ := ChipMem-Obergrenze - Stacklänge
Ø3CØ         sub.l    aØ,dØ               ; - Ende Exec-Datenbereich
Ø3C2         bsr      FC19EA              ; ---> AddMemList
Ø3C6         movea.l  a6,a1               ; a1 := SysBase
Ø3C8         bsr      FC14ØC              ; ---> AddLibrary
Ø3CC         lea      3AA(pc),aØ          ; aØ := FCØ778 = Ausn.-Vektoren-Tabelle
Ø3DØ         movea.l  aØ,a1               ; a1 := aØ
Ø3D2         movea.w  #8,a2               ; a2 := Adresse des ersten Vektors
Ø3D6         bra.s    FCØ3DE              ; --->
Ø3D8
Ø3D8 FCØ3D8  lea      Ø(aØ,dØ.w),a3       ; a3 := Ausnahmevektor
Ø3DC         move.l   a3,(a2)+            ; in Vektorbereich schreiben
Ø3DE FCØ3DE  move.w   (a1)+,dØ            ; dØ := Offset aus Tabelle
Ø3EØ         bne.s    FCØ3D8              ; Tabelle nicht zu Ende: ->
Ø3E2         move.w   128(a6),dØ          ; dØ := AttnFlags
Ø3E6         btst.l   #Ø,dØ               ; Bit Ø gesetzt?
Ø3EA         beq.s    FCØ41E              ; nein: Prozessor MC68ØØØ ->
Ø3EC         lea      48E(pc),aØ          ; aØ := FCØ87C
Ø3FØ         movea.w  #8,a1               ; a1 -> Busfehler-Vektor
Ø3F4         move.l   aØ,(a1)+            ; Busfehler-Vektor
Ø3F6         move.l   aØ,(a1)+            ; Adressfehler-Vektor
Ø3F8         move.l   #FCØ8BA,-1C(a6)     ; Supervisor-Aufruf
Ø4ØØ         move.l   #42CØ4E75,-21Ø(a6)  ; GetCC := move ccr,dØ - rts
Ø4Ø8         btst.l   #4,dØ               ; MC 68881 im System?
Ø4ØC         beq.s    FCØ41E              ; nein: ->
Ø4ØE         move.l   #FC1Ø8A,-34(a6)     ; Switch-Aufruf ändern
Ø416         move.l   #FC1ØE8,-3A(a6)     ; Dispatch-Aufruf ändern
Ø41E FCØ41E  bsr      FC125C              ; ---> Int Server initialisieren
Ø422         lea      DFFØØØ,aØ           ; aØ := Spezialchip-Basisadresse
Ø428         move.w   #82ØØ,96(aØ)        ; DMACON: DMA Ø bis 8 freigeben
Ø42E         move.w   #CØØØ,9A(aØ)        ; INTENA: Interrupts freigeben
Ø434         move.w   #FFFF,126(a6)       ; Forbid-, Disable-Ebene rücksetzen
Ø43A         bsr      FC22FA              ; ---> ROMWack initialisieren
```

```
Ø43E        moveq    #Ø,d1            ; dØ als Summenregister löschen
Ø44Ø        lea      22(a6),aØ        ; aØ -> Anfang des Prüfsummenbereichs
Ø444        move.w   #16,dØ           ; dØ := 22: Wortzähler
Ø448 FCØ448 add.w    (aØ)+,d1         ; 23 Worte in d1 addieren
Ø44A        dbra     dØ,FCØ448
Ø44E        not.w    d1               ; Ergebnis invertieren
Ø45Ø        move.w   d1,52(a6)        ; und als ChkSum speichern
Ø454
Ø454 ;------ Exec als Dauertask einrichten
Ø454
Ø454        lea      76(pc),aØ        ; aØ := FCØ4CC: Tabelle f. AllocEntry
Ø458        bsr      FC191E           ; ---> AllocEntry
Ø45C        movea.l dØ,a2             ; a2 -> Anfang des zugeordn. Speichers
Ø45E        lea      1Ø1Ø(a2),aØ      ; aØ := a2 + $1Ø1Ø
Ø462        lea      8(aØ),a1         ; a1 := aØ + 8 -> Task Control Struct
Ø466        addi.l   #1Ø,dØ
Ø46C        move.l   dØ,3A(a1)        ; tc_SPLower
Ø47Ø        move.l   aØ,3E(a1)        ; tc_SPUpper
Ø474        move.l   aØ,36(a1)        ; tc_SPReg
Ø478        move     aØ,usp           ; User-Stackpointer := aØ
Ø47A        clr.b    9(a1)            ; ln_Pri
Ø47E        move.b   1,8(a1)          ; ln_Type (sollte wohl '#1' sein!)
Ø484        move.l   #FCØØA8,A(a1)    ; ln_Name -> 'exec.library'
Ø48C        lea      4A(a1),aØ        ; aØ -> tc_MemEntry
Ø49Ø        move.l   aØ,(aØ)          ; List Header initialisieren
Ø492        addq.l   #4,(aØ)
Ø494        clr.l    4(aØ)
Ø498        move.l   aØ,8(aØ)
Ø49C        exg      a2,a1            ; a1 -> Anfang des zugeordn. Speichers
Ø49E        bsr      FC15D8           ; ---> AddHead
Ø4A2        exg      a2,a1            ; a1, a2 zurücktauschen
Ø4A4        move.l   a1,114(a6)       ; ThisTask := a1
Ø4A8        suba.l   a2,a2            ; a2 := Ø
Ø4AA        movea.l a2,a3             ; a3 := Ø
Ø4AC        bsr      FC1C48           ; ---> AddTask
Ø4BØ        movea.l 114(a6),a1        ; a1 -> Task Control Structure
Ø4B4        move.b   #2,F(a1)         ; tc_State := RUN
Ø4BA        bsr      FC16ØØ           ; ---> Remove
Ø4BE        andi.w   #Ø,sr            ; Supervisor-Status löschen
Ø4C2        addq.b   #1,127(a6)       ; Forbid
```

```
04C6         jsr      -8A(a6)           ; ---> Permit
04CA         bra.s    FC0500            ; --->
04CC
04CC ;------ Tabelle für Speicheranforderung für Exec-Task
04CC
04CC FC04CC  DC.L     0                 ; 14 Bytes reserviert für
04D0         DC.L     0                 ; List Node
04D4         DC.L     0
04D8         DC.W     0
04DA         DC.W     1                 ; ml_NumEntries: Zahl der Einträge
04DC         DC.L     10001             ; me_Reqs: Public, Clear
04E0         DC.L     1064              ; me_Length: Angeforderte Bytezahl
04E4
04E4 ;------ Tabelle für Suche nach residenten Moduln
04E4
04E4 FC04E4  DC.L      FC0000           ; Speicherbereich 1
04E8         DC.L     1000000
04EC         DC.L      FC0000           ; Speicherbereich 2
04F0         DC.L     1000000
04F4         DC.L      F00000           ; Speicherbereich 3
04F8         DC.L      F80000
04FC         DC.L     -1
0500
0500 ;------ Alle residenten Moduln suchen und initialisieren
0500
0500 FC0500  lea      -1E(pc),a0        ; a0 := FC04E4 = Suchtabelle
0504         bsr      FC0900            ; ---> Resident-Liste erzeugen
0508         move.l   d0,12C(a6)        ; Zeiger auf Resident-Liste
050C
050C         bclr.b   #1,BFE001         ; 8520-A PRA: LED hell
0514         move.l   2E(a6),d0         ; d0 := CoolCapture-Vektor
0518         beq.s    FC051E            ; nicht vorhanden: ->
051A         movea.l  d0,a0             ; a0 := CoolCapture-Vektor
051C         jsr      (a0)              ; ---> CoolCapture-Routine
051E FC051E  moveq    #1,d0             ; rt_Flag := Kaltstart
0520         moveq    #0,d1             ; Version := 0
0522         bsr      FC0AF0            ; ---> InitCode (startet DOS)
0526         move.l   32(a6),d0         ; d0 := WarmCapture-Vektor
052A         beq.s    FC0530            ; nicht vorhanden: ->
052C         movea.l  d0,a0             ; a0 := WarmCapture-Vektor
```

```
052E          jsr      (a0)                 ; ---> WarmCapture-Routine
0530 FC0530   moveq    #D,d0                ; d0 := 13 (Zähler)
0532 FC0532   clr.l    -(a7)                ; 14 Langworte im Stack löschen
0534          dbra     d0,FC0532
0538          movem.l (a7)+,d0-d7/a0-a5    ; Register löschen
053C FC053C   jsr      -72(a6)              ; ---> Debug
0540          movea.l 4,a6                 ; a6 := SysBase
0544          bra.s    FC053C               ; ---> Loop
0546
0546 ;------ Prozessor-Typ ermitteln
0546
0546 FC0546   movem.l a2-a3,-(a7)          ; Register retten
054A          movea.l 10,a0                ; a0 := Trapvektor "Illegaler Befehl"
054E          movea.l 2C,a2                ; a2 := Trapvektor "1111"
0552          lea      2E(pc),a1            ; a1 := FC0582
0556          move.l   a1,10                ; Trap "Illegaler Befehl" setzen
055A          move.l   a1,2C                ; Trap "1111" setzen
055E          movea.l a7,a1                ; a1 := SSP
0560          moveq    #0,d0                ; AttnFlags rücksetzen
0562          moveq    #0,d1                ; d1 := 0
0564          rte
0566
0566          move.b   d1,d4
0568          bset.l   #0,d0                ; AttnFlags Bit 0: M 68010
056C          moveq    #1,d1                ; d1 := 1
056E          rte
0570
0570          move.b   d2,d0
0572          bset.l   #1,d0                ; AttnFlags Bit 1: M 68020
0576          cpgen    ... d1
0578
057A          tst.l    d1
057C          bne.s    FC0582
057E          bset.l   #4,d0                ; AttnFlags Bit 4: M 68881
0582 FC0582   movea.l a1,a7                ; a7 wiederherstellen
0584          move.l   a0,10                ; Trap "Illegaler Befehl" und
0588          move.l   a2,2C                ; Trap "1111" wiederherstellen
058C          movem.l (a7)+,a2-a3          ; a2 und a3 wiederherstellen
0590          rts
0592
```

```
0592 ;------ Speichergrenze ermitteln
0592
0592 FC0592  moveq    #0,d1                ; d1 := 0
0594         move.l   d1,(a0)              ; Erste Adresse mit 0 belegen
0596         movea.l  a0,a2                ; Anfangsadresse in a2 retten
0598         move.l   #F2D4B698,d0         ; Bitmuster für Test
059E FC059E  lea      1000(a0),a0          ; a0 um 4 kB erhöhen
05A2         cmpa.l   a0,a1                ; Endadresse erreicht?
05A4         bls.s    FC05B0               ; ja: ->
05A6         move.l   d0,(a0)              ; Bitmuster abspeichern
05A8         tst.l    (a2)                 ; Erste Zelle im Bereich = 0?
05AA         bne.s    FC05B0               ; nein: erster Durchlauf beendet ->
05AC         cmp.l    (a0),d0              ; richtiges Bitmuster im Speicher?
05AE         beq.s    FC059E               ; ja: weitermachen ->
05B0 FC05B0  movea.l  a0,a3                ; a3 := letzte RAM-Adresse + 1
05B2         jmp      (a5)                 ; ---> Rücksprung
05B4
05B4 ;------ Fehlerausgang
05B4
05B4 FC05B4  move.w   #CC0,d0              ; Farbkode: Rot+Grün
05B8 FC05B8  lea      DFF000,a4            ; a4 := Spezialchip-Basisadresse
05BE         move.w   #200,100(a4)         ; BPLCON0: Comp. Video Color Enable
05C4         move.w   #0,110(a4)           ; BPL1DAT: Bildschirmausgabe triggern
05CA         move.w   d0,180(a4)           ; COLOR0: Hintergrundfarbe setzen
05CE         moveq    #A,d1                ; d1 := 10 (Zähler 1)
05D0         moveq    #FF,d0               ; d0 := 65535 (Zähler 2)
05D2 FC05D2  bset.b   #1,BFE001            ; 8520-A PRA: LED dunkel
05DA         dbra     d0,FC05D2            ; ---> ca. 0,15 s warten
05DE         lsr.w    #1,d0                ; d0 := 32767
05E0 FC05E0  bclr.b   #1,BFE001            ; 8520-A PRA: LED hell
05E8         dbra     d0,FC05E0            ; ---> ca. 0,08 s warten
05EC         dbra     d1,FC05D2            ; ---> Blinken 10 mal wiederholen
05F0
05F0 ;------ boot / ig
05F0
05F0 FC05F0  move.l   #20000,d0            ; d0 := 131072 (Zähler)
05F6 FC05F6  subq.l   #1,d0                ; Warteschleife ca. 0,3 s
05F8         bgt.s    FC05F6
05FA         reset                         ; Boot-ROM einschalten
05FC         movea.l  4,a0                 ; a0 := Startadresse
```

```
0600        jmp     (a0)                ; ---> Neustart
0602
0602 ;------ Speicher löschen
0602
0602 FC0602 moveq   #0,d2               ; d2 := Löschkode
0604        sub.l   a0,d0               ; d0 := Länge des Löschbereichs
0606        lsr.l   #2,d0               ; durch 4: Anzahl der Langworte
0608        move.l  d0,d1               ; nach d1 kopieren
060A        swap    d1                  ; oberen Teil nach unten holen
060C        bra.s   FC0610              ; --->
060E
060E FC060E move.l  d2,(a0)+            ; Löschkode speichern
0610 FC0610 dbra    d0,FC060E           ; wiederholen, bis d0 = 0
0614        dbra    d1,FC060E           ; wiederholen, bis d1 = 0
0618        jmp     (a5)                ; ---> Rücksprung
061A
061A ;------ RAM im Bereich ab C00000 suchen
061A
061A FC061A movea.l a0,a4               ; a4 := a0 = Suchbereich Anfang
061C FC061C movea.l a4,a2               ; a2 := a4
061E        adda.l  #40000,a2           ; 256 kB addieren
0624        move.w  #3FFF,INTENA-1000(a2)  ; Wenn kein RAM vorhanden ist, wird
062A        tst.w   INTENAR-1000(a2)       ; der Bereich ab DFF000 abgebildet
062E        bne.s   FC063E                 ; Abbildung liegt nicht vor: ->
0630        move.w  #BFFF,INTENA-1000(a2)  ; sonst alle Bits setzen
0636        cmpi.w  #3FFF,INTENAR-1000(a2) ; sind sie gesetzt?
063C        beq.s   FC0644              ; ja: Abbildung liegt vor ->
063E FC063E movea.l a2,a4               ; a4 := a2
0640        cmpa.l  a4,a1               ; a4 < Suchbereich Ende?
0642        bhi.s   FC061C              ; ja: Suche fortsetzen ->
0644 FC0644 move.w  #7FFF,INTENA-1000(a2)  ; Alle Interrupts sperren
064A        cmpa.l  a0,a4               ; Suchadresse = Suchbereich Anfang?
064C        bne.s   FC0650              ; nein: ->
064E        suba.l  a4,a4               ; a4 := 0 ('kein RAM gefunden')
0650 FC0650 jmp     (a5)                ; ---> Rücksprung
0652
0652        DC.W    0
0654
0654 ;----------------------------------    AddDevice
0654
```

```
0654 FC0654  lea      15E(a6),a0        ; a0 := Device List Header Adresse
0658         bsr      FC1682            ; ---> Forbid, Enqueue, Permit
065C         jsr      -1AA(a6)          ; ---> SumLibrary
0660         rts
0662
0662 ;--------------------------------   RemDevice
0662
0662 FC0662  bra      FC141A            ; ---> RemLibrary
0666
0666 ;--------------------------------   OpenDevice
0666
0666 FC0666  move.l   a2,-(a7)          ; a2 retten
0668         movea.l  a1,a2             ; a2 := a1 -> IORequest
066A         clr.b    1F(a1)            ; io_Error löschen
066E         movem.l  d0-d1,-(a7)       ; d0 und d1 retten
0672         movea.l  a0,a1             ; a1 := a0 -> Device-Name
0674         lea      15E(a6),a0        ; a0 := Device List Header Adresse
0678         addq.b   #1,127(a6)        ; Forbid
067C         bsr      FC165A            ; ---> FindName
0680         movea.l  d0,a0             ; a0 := d0 -> Node
0682         movem.l  (a7)+,d0-d1       ; d0 und d1 wiederherstellen
0686         move.l   a0,14(a2)         ; *io-Device in IORequest eintragen
068A         beq.s    FC06AC            ; Namen nicht gefunden: ->
068C         clr.l    18(a2)            ; *io_Unit in IORequest löschen
0690         movea.l  a2,a1             ; a1 := a2 -> IORequest
0692         move.l   a6,-(a7)          ; a6 retten
0694         movea.l  a0,a6             ; a6 := a0 -> Device Node
0696         jsr      -6(a6)            ; ---> Open Device
069A         movea.l  (a7)+,a6          ; a6 wiederherstellen
069C         move.b   1F(a2),d0         ; d0 := io_Error
06A0         ext.w    d0                ; auf Wort
06A2         ext.l    d0                ; und Langwort erweitern
06A4 FC06A4  jsr      -8A(a6)           ; ---> Permit
06A8         movea.l  (a7)+,a2          ; a2 wiederherstellen
06AA         rts
06AC
06AC FC06AC  moveq    #FF,d0            ; d0 := -1
06AE         move.b   d0,1F(a2)         ; io_Error := -1
06B2         bra.s    FC06A4            ; --->
06B4
```

```
Ø6B4 ;---------------------------------  CloseDevice
Ø6B4
Ø6B4 FCØ6B4  addq.b  #1,127(a6)        ; Forbid
Ø6B8         move.l  a6,-(a7)          ; a6 retten
Ø6BA         movea.l 14(a1),a6         ; a6 -> io_Device
Ø6BE         jsr     -C(a6)            ; ---> Close Device
Ø6C2         movea.l (a7)+,a6          ; a6 wiederherstellen
Ø6C4         jsr     -8A(a6)           ; ---> Permit
Ø6C8         rts
Ø6CA
Ø6CA ;---------------------------------  SendIO
Ø6CA
Ø6CA FCØ6CA  clr.b   1E(a1)            ; io_Flags := Ø
Ø6CE         move.l  a6,-(a7)          ; a6 retten
Ø6DØ         movea.l 14(a1),a6         ; a6 -> io_Device
Ø6D4         jsr     -1E(a6)           ; ---> BeginIO
Ø6D8         movea.l (a7)+,a6          ; a6 wiederherstellen
Ø6DA         rts
Ø6DC
Ø6DC ;---------------------------------  DoIO
Ø6DC
Ø6DC FCØ6DC  move.l  a1,-(a7)          ; a1 retten
Ø6DE         move.b  #1,1E(a1)         ; io_Flags := 1
Ø6E4         move.l  a6,-(a7)          ; a6 retten
Ø6E6         movea.l 14(a1),a6         ; a6 -> io_Device
Ø6EA         jsr     -1E(a6)           ; ---> BeginIO
Ø6EE         movea.l (a7)+,a6          ; a6 wiederherstellen
Ø6FØ         movea.l (a7)+,a1          ; a1 wiederherstellen
Ø6F2
Ø6F2 ;---------------------------------  WaitIO
Ø6F2
Ø6F2 FCØ6F2  btst.b  #Ø,1E(a1)         ; io_Flags = Ø?
Ø6F8         bne.s   FCØ744            ; nein: ->
Ø6FA         move.l  a2,-(a7)          ; a2 retten
Ø6FC         movea.l a1,a2             ; a2 := a1 -> IORequest
Ø6FE         movea.l E(a2),aØ          ; aØ := mn_ReplyPort
Ø7Ø2         move.b  F(aØ),d1          ; d1 := mp_Flags
Ø7Ø6         moveq   #Ø,dØ             ; dØ := Ø
Ø7Ø8         bset.l  d1,dØ             ; mp_Flags-Bit setzen
Ø7ØA         move.w  #4ØØØ,INTENA      ; alle Interrupts abschalten
```

```
0712         addq.b  #1,126(a6)          ; Disable
0716 FC0716  cmpi.b  #7,8(a2)            ; ln_Type = nt_ReplyMsg?
071C         beq.s   FC0724              ; ja: ->
071E         jsr     -13E(a6)            ; ---> Wait
0722         bra.s   FC0716              ; ---> Loop
0724
0724 FC0724  movea.l a2,a1               ; a1 := a2 -> IORequest
0726         movea.l (a1),a0             ; Remove Node
0728         movea.l 4(a1),a1
0728         movea.l 4(a1),a1
072C         move.l  a0,(a1)
072E         move.l  a1,4(a0)
0732         subq.b  #1,126(a6)          ; Enable
0736         bge.s   FC0740              ; noch keine Freigabe: ->
0738         move.w  #C000,INTENA        ; Interrupts freigeben
0740 FC0740  movea.l a2,a1               ; a1 := a2 -> IORequest
0742         movea.l (a7)+,a2            ; a2 wiederherstellen
0744 FC0744  move.b  1F(a1),d0           ; d0 := io_Error
0748         ext.w   d0                  ; auf Wort
074A         ext.l   d0                  ; und Langwort erweitern
074C         rts
074E
074E ;--------------------------------- CheckIO
074E
074E FC074E  btst.b  #0,1E(a1)           ; io_Error Bit 0 gesetzt?
0754         beq.s   FC075A              ; nein: ->
0756         move.l  a1,d0               ; d0 := TRUE
0758         rts
075A
075A FC075A  cmpi.b  #7,8(a1)            ; nt_ReplyMsg?
0760         beq.s   FC0766              ; ja: ->
0762         moveq   #0,d0               ; d0 := FALSE
0764         rts
0766
0766 FC0766  move.l  a1,d0               ; d0 := TRUE
0768         rts
076A
076A ;--------------------------------- AbortIO
076A
076A FC076A  move.l  a6,-(a7)            ; a6 retten
```

```
Ø76C        movea.l 14(a1),a6         ; a6 -> io_Device
Ø77Ø        jsr     -24(a6)           ; ---> AbortIO
Ø774        movea.l (a7)+,a6          ; a6 wiederherstellen
Ø776        rts
Ø778
Ø778 ;---------------------------------- Offsets der Ausnahme-Vektoren
Ø778
Ø778 FCØ778 DC.W    ØØ64              ; FCØ7DC      VØ2:  Busfehler
Ø77A        DC.W    ØØ66              ; FCØ7DE      VØ3:  Adressfehler
Ø77C        DC.W    ØØ68              ; FCØ7EØ      VØ4:  Illegaler Befehl
Ø77E        DC.W    ØØ6A              ; FCØ7E2      VØ5:  Division durch Null
Ø78Ø        DC.W    ØØ6C              ; FCØ7E4      VØ6:  CHK
Ø782        DC.W    ØØ6E              ; FCØ7E6      VØ7:  TRAPV
Ø784        DC.W    Ø15A              ; FCØ8D2      VØ8:  Privilegverletzung
Ø786        DC.W    ØØ72              ; FCØ7EA      VØ9:  TRACE
Ø788        DC.W    ØØ74              ; FCØ7EC      VØA:  1Ø1Ø
Ø78A        DC.W    ØØ76              ; FCØ7EE      VØB:  1111
Ø78C        DC.W    ØØ78              ; FCØ7FØ      VØC:  reserviert
Ø78E        DC.W    ØØ7A              ; FCØ7F2      VØD:  reserviert
Ø79Ø        DC.W    ØØ7C              ; FCØ7F4      VØE:  reserviert
Ø792        DC.W    ØØ7E              ; FCØ7F6      VØF:  n. initialis. Interr.
Ø794        DC.W    ØØ8Ø              ; FCØ7F8      V1Ø:  reserviert
Ø796        DC.W    ØØ8Ø              ; FCØ7F8      V11:  reserviert
Ø798        DC.W    ØØ8Ø              ; FCØ7F8      V12:  reserviert
Ø79A        DC.W    ØØ8Ø              ; FCØ7F8      V13:  reserviert
Ø79C        DC.W    ØØ8Ø              ; FCØ7F8      V14:  reserviert
Ø79E        DC.W    ØØ8Ø              ; FCØ7F8      V15:  reserviert
Ø7AØ        DC.W    ØØ8Ø              ; FCØ7F8      V16:  reserviert
Ø7A2        DC.W    ØØ8Ø              ; FCØ7F8      V17:  reserviert
Ø7A4        DC.W    ØØ8Ø              ; FCØ7F8      V18:  reserviert
Ø7A6        DC.W    Ø4DA              ; FCØC52      V19:  Interrupt 1
Ø7A8        DC.W    Ø52E              ; FCØCA6      V1A:  Interrupt 2
Ø7AA        DC.W    Ø56Ø              ; FCØCD8      V1B:  Interrupt 3
Ø7AC        DC.W    Ø5B8              ; FCØD3Ø      V1C:  Interrupt 4
Ø7AE        DC.W    Ø646              ; FCØDBE      V1D:  Interrupt 5
Ø7BØ        DC.W    Ø68C              ; FCØEØ4      V1E:  Interrupt 6
Ø7B2        DC.W    Ø6D2              ; FCØE4A      V1F:  Interrupt 7
Ø7B4        DC.W    ØØ82              ; FCØ7FA      V2Ø:  Trap #Ø
Ø7B6        DC.W    ØØ84              ; FCØ7FC      V21:  Trap #1
Ø7B8        DC.W    ØØ86              ; FCØ7FE      V22:  Trap #2
```

```
Ø7BA        DC.W   ØØ88            ; FCØ8ØØ      V23:  Trap #3
Ø7BC FCØ7BC DC.W   ØØ8A            ; FCØ8Ø2      V24:  Trap #4
Ø7BE        DC.W   ØØ8C            ; FCØ8Ø4      V25:  Trap #5
Ø7CØ        DC.W   ØØ8E            ; FCØ8Ø6      V26:  Trap #6
Ø7C2        DC.W   ØØ9Ø            ; FCØ8Ø8      V27:  Trap #7
Ø7C4        DC.W   ØØ92            ; FCØ8ØA      V28:  Trap #8
Ø7C6        DC.W   ØØ94            ; FCØ8ØC      V29:  Trap #9
Ø7C8        DC.W   ØØ96            ; FCØ8ØE      V2A:  Trap #A
Ø7CA        DC.W   ØØ98            ; FCØ81Ø      V2B:  Trap #B
Ø7CC        DC.W   ØØ9A            ; FCØ812      V2C:  Trap #C
Ø7CE        DC.W   ØØ9C            ; FCØ814      V2D:  Trap #D
Ø7DØ        DC.W   ØØ9E            ; FCØ816      V2E:  Trap #E
Ø7D2        DC.W   ØØAØ            ; FCØ818      V2F:  Trap #F
Ø7D4        DC.W   Ø
Ø7D6        DC.W   Ø
Ø7D8
Ø7D8 ;------ Ausnahme-Aufrufe
Ø7D8
Ø7D8        bsr.s  FCØ828          ; VØØ
Ø7DA FCØ7DA bsr.s  FCØ828          ; VØ1
Ø7DC FCØ7DC bsr.s  FCØ83A          ; VØ2
Ø7DE FCØ7DE bsr.s  FCØ83A          ; VØ3
Ø7EØ FCØ7EØ bsr.s  FCØ85Ø          ; VØ4
Ø7E2 FCØ7E2 bsr.s  FCØ85Ø          ; VØ5
Ø7E4 FCØ7E4 bsr.s  FCØ85Ø          ; VØ6
Ø7E6 FCØ7E6 bsr.s  FCØ85Ø          ; VØ7
Ø7E8        bsr.s  FCØ85Ø          ; VØ8
Ø7EA FCØ7EA bsr.s  FCØ85Ø          ; VØ9
Ø7EC FCØ7EC bsr.s  FCØ85Ø          ; VØA
Ø7EE FCØ7EE bsr.s  FCØ85Ø          ; VØB
Ø7FØ FCØ7FØ bsr.s  FCØ85Ø          ; VØC
Ø7F2 FCØ7F2 bsr.s  FCØ85Ø          ; VØD
Ø7F4 FCØ7F4 bsr.s  FCØ828          ; VØE
Ø7F6 FCØ7F6 bsr.s  FCØ828          ; VØF
Ø7F8 FCØ7F8 bra.s  FCØ81A          ; V1Ø,...,V18
Ø7FA
Ø7FA FCØ7FA bsr.s  FCØ866          ; V2Ø
Ø7FC FCØ7FC bsr.s  FCØ866          ; V21
Ø7FE FCØ7FE bsr.s  FCØ866          ; V22
Ø8ØØ FCØ8ØØ bsr.s  FCØ866          ; V23
```

```
0802 FC0802  bsr.s   FC0866               ; V24
0804 FC0804  bsr.s   FC0866               ; V25
0806 FC0806  bsr.s   FC0866               ; V26
0808 FC0808  bsr.s   FC0866               ; V27
080A FC080A  bsr.s   FC0866               ; V28
080C FC080C  bsr.s   FC0866               ; V29
080E FC080E  bsr.s   FC0866               ; V2A
0810 FC0810  bsr.s   FC0866               ; V2B
0812 FC0812  bsr.s   FC0866               ; V2C
0814 FC0814  bsr.s   FC0866               ; V2D
0816 FC0816  bsr.s   FC0866               ; V2E
0818 FC0818  bsr.s   FC0866               ; V2F
081A
081A ;------ V10 bis V18
081A
081A FC081A  ori.w   #700,sr              ; IR-Ebene 7 setzen
081E         move.l  #8100000A,-(a7)      ; Alert-Kode 'AN_BogusExcpt'
0824         bra     FC2FB4               ; ---> Alert
0828
0828 ;------ V0E, V0F
0828
0828 FC0828  ori.w   #700,sr              ; IR-Ebene 7 setzen
082C         subi.l  #FC07DA,(a7)         ; Vektornummer erzeugen
0832         lsr.w   2(a7)
0836         bra     FC2FB4               ; ---> Alert
083A
083A ;------ V02, V03: Busfehler, Adressfehler
083A
083A FC083A  subi.l  #FC07DA,(a7)         ; Vektornummer erzeugen
0840         lsr.w   2(a7)
0844         btst.b  #5,C(a7)             ; Supervisor-Modus?
084A         beq.s   FC0894               ; nein: ->
084C         bra     FC2FB4               ; ---> Alert
0850
0850 ;------ V08 bis V0D
0850
0850 FC0850  subi.l  #FC07DA,(a7)         ; Vektornummer erzeugen
0856         lsr.w   2(a7)
085A FC085A  btst.b  #5,4(a7)             ; Supervisor-Modus?
0860         beq.s   FC0894               ; nein: ->
```

```
0862         bra      FC2FB4              ; ---> Alert
0866
0866 ;------ TRAP-Befehl
0866
0866 FC0866  subi.l   #FC07BC,(a7)        ; Vektornummer erzeugen
086C         lsr.w    2(a7)
0870         btst.b   #5,4(a7)            ; Supervisor-Modus?
0876         bne      FC2FB4              ; ja: Alert ->
087A         bra.s    FC0894              ; --->
087C
087C ;------ Busfehler / Adressfehler bei M 68010/68020
087C
087C FC087C  clr.l    -(a7)               ; Langwort im Stack löschen
087E         move.w   A(a7),2(a7)         ; Vektor-Offset eintragen
0884         andi.w   #FFF,2(a7)          ; und isolieren
088A         lsr.w    2(a7)               ; durch 4 dividieren
088E         lsr.w    2(a7)               ; ergibt die Vektor-Nummer
0892         bra.s    FC085A              ; --->
0894
0894 ;------ Ausgang für Task-Trap
0894
0894 FC0894  movem.l a0-a1,-(a7)
0898         movea.l 4,a0                 ; a0 := SysBase
089C         movea.l 114(a0),a0           ; a0 := ThisTask
08A0         move.l   32(a0),4(a7)        ; tc_TrapCode-Vektor in Stack
08A6         movea.l (a7)+,a0             ; a0 vom Stack nehmen
08A8         rts                          ; ---> TaskTrap-Routine
08AA
08AA ;----------------------------------  Supervisor
08AA
08AA FC08AA  ori.w    #2000,sr            ; Supervisor-Flag setzen
08AE         pea      FC08B8              ; Rückkehradresse auf Stack
08B4         move     sr,-(a7)            ; Statusregister auf Stack
08B6         jmp      (a5)                ; --->
08B8
08B8 FC08B8  rts
08BA
08BA ;------ Supervisor für M 68010/68020
08BA
08BA FC08BA  ori.w    #2000,sr            ; Supervisor-Flag setzen
```

```
08BE         subq.l  #8,a7            ; Platz für 8 Bytes im Stack schaffen
08C0         move    sr,(a7)          ; Statusregister auf Stack
08C2         move.l  #FC08B8,2(a7)    ; Rückkehradresse auf Stack
08CA         move.w  #20,6(a7)        ; Formatkode 0, Vektoroffset auf Stack
08D0         jmp     (a5)             ; --->
08D2
08D2 ;------ Privilegverletzung
08D2
08D2 FC08D2  cmpi.l  #FC08AA,2(a7)    ; erzeugt durch 'Supervisor 68000'?
08DA         beq.s   FC08E6           ; ja: ->
08DC         cmpi.l  #FC08BA,2(a7)    ; erzeugt durch 'Supervisor 68010+'?
08E4         bne.s   FC08F0           ; nein: ->
08E6 FC08E6  move.l  #FC08B8,2(a7)    ; Rückkehradresse auf Stack
08EE         jmp     (a5)             ; --->
08F0
08F0 FC08F0  ori.w   #700,sr          ; IR-Ebene 7 setzen
08F4         move.l  #8,-(a7)         ; Vektornummer auf Stack
08FA         bra     FC085A           ; --->
08FE
08FE         DC.W    0
0900
0900 ;------ Liste aller residenten Moduln erzeugen
0900
0900 FC0900  movem.l d3-d4/a2-a4,-(a7) ; Register retten
0904         link    a5,#-E           ; Platz für List Header schaffen
0908         movea.l a7,a3            ; a3 := a7 -> List Header
090A         move.l  a3,(a3)          ; List Header initialisieren
090C         addq.l  #4,(a3)
090E         clr.l   4(a3)
0912         move.l  a3,8(a3)
0916         movea.l a0,a2            ; a2 := a0 -> Suchbereichs-Tabelle
0918 FC0918  tst.l   (a2)             ; Langwort in Tabelle
091A         bmi.s   FC0926           ; Tabellenende: ->
091C         movea.l (a2)+,a4         ; a4 -> Suchbereich-Anfang
091E         move.l  (a2)+,d4         ; d4 -> Suchbereich-Ende
0920         bsr     FC0948           ; ---> Bereich durchsuchen
0924         bra.s   FC0918           ; ---> Loop
0926
0926 FC0926  bsr     FC0A3C           ; ---> SumKickData
092A         cmp.l   22A(a6),d0       ; d0 = KickChkSum?
```

```
092E          bne.s    FC093C          ; nein: ->
0930          bsr      FC0A94          ; ---> Speicherplatz reservieren
0934          tst.l    d0              ; d0 = 0?
0936          beq.s    FC093C          ; ja: ->
0938          bsr      FC0A14          ; --->
093C FC093C   bsr      FC09DE          ; ---> Liste anlegen
0940          unlk     a5              ; List Header wieder freigeben
0942          movem.l  (a7)+,d3-d4/a2-a4 ; Register wiederherstellen
0946          rts
0948
0948 FC0948   movem.l  d2/a5,-(a7)     ; Register retten
094C          move.w   #4AFC,d2        ; d2 := Kennwort ('Illegaler Befehl')
0950 FC0950   move.l   d4,d0           ; d0 := d4 = Suchbereich-Ende
0952          sub.l    a4,d0           ; d0 := a4-d0 = Suchbereich-Länge
0954          bls.s    FC097E          ; Suchbereich leer: ->
0956          lsr.l    #1,d0           ; d0 := d0/2 = Länge in Worten
0958          subq.l   #1,d0           ; d0 := d0-1 = Wortzähler
095A          move.l   d0,d1           ; d0 nach d1 kopieren
095C          swap     d1              ; d1.w := H-Wort des Wortzählers
095E          bra.s    FC0962          ; --->
0960
0960 FC0960   cmp.w    (a4)+,d2        ; Kennwort mit Speicherwort vergleichen
0962 FC0962   dbeq     d0,FC0960       ; bis Gleichheit festgestellt wird
0966          dbeq     d1,FC0960       ; oder Suchbereich zu Ende
096A          bne.s    FC097E          ; Suchbereich zu Ende: ->
096C          lea      -2(a4),a5       ; a5 -> Kennwort
0970          cmpa.l   (a4),a5         ; Folgt dem Kennwort seine Adresse?
0972          bne.s    FC0962          ; nein: weitersuchen ->
0974          bsr      FC0984          ; ---> Node in Liste einfügen
0978          movea.l  6(a5),a4        ; a4 -> Ende des residenten Moduls
097C          bra.s    FC0950          ; ---> Loop
097E
097E FC097E   movem.l  (a7)+,d2/a5     ; Register wiederherstellen
0982          rts
0984
0984 FC0984   movea.l  a3,a0           ; a0 := a3 -> List Header
0986          movea.l  E(a5),a1        ; a1 -> rt_LibName
098A          bsr      FC165A          ; ---> FindName
098E          tst.l    d0              ; Name schon in der Liste?
0990          beq.s    FC09B8          ; nein: neuen Node erzeugen ->
```

```
Ø992            movea.l dØ,a1           ; a1 := dØ -> Node mit dem Namen
Ø994            movea.l E(a1),aØ        ; aØ -> Struktur mit dem Namen
Ø998            move.b  B(a5),dØ        ; dØ := gerade gefundene Version
Ø99C            cmp.b   B(aØ),dØ        ; vergleichen mit früher gefundenen
Ø9AØ            blt.s   FCØ9DC          ; gerade gefundene kleiner: ->
Ø9A2            bgt.s   FCØ9AE          ; früher gefundene kleiner: ->
Ø9A4            move.b  D(a5),dØ        ; dØ := gerade gefundene Prioriràt
Ø9A8            cmp.b   D(aØ),dØ        ; vergleichen mit früher gefundenen
Ø9AC            blt.s   FCØ9DC          ; gerade gefundene kleiner: ->
Ø9AE FCØ9AE     move.l  a1,dØ           ; a1 in dØ retten
Ø9BØ            bsr     FC16ØØ          ; ---> Remove (früher gefundenen)
Ø9B4            movea.l dØ,a1           ; a1 wiederherstellen
Ø9B6            bra.s   FCØ9C6          ; ---> neuen Node einfügen
Ø9B8
Ø9B8 FCØ9B8     moveq   #Ø,d1           ; d1 := Speichertyp: beliebig
Ø9BA            moveq   #12,dØ          ; dØ := Zahl der benötigten Bytes
Ø9BC            jsr     -C6(a6)         ; ---> AllocMem
Ø9CØ            tst.l   dØ              ; Speicher bereitgestellt?
Ø9C2            beq.s   FCØ9DC          ; nein: ->
Ø9C4            movea.l dØ,a1           ; a1 := dØ -> neuer Node
Ø9C6 FCØ9C6     move.b  D(a5),9(a1)     ; Priorität eintragen
Ø9CC            move.l  E(a5),A(a1)     ; Zeiger auf Namen eintragen
Ø9D2            move.l  a5,E(a1)        ; Zeiger auf Struktur eintragen
Ø9D6            movea.l a3,aØ           ; aØ := a3 -> List Header
Ø9D8            bsr     FC1634          ; ---> Enqueue
Ø9DC FCØ9DC     rts
Ø9DE
Ø9DE FCØ9DE     moveq   #4,dØ           ; dØ als Bytezähler initialisieren
Ø9EØ            move.l  (a3),d4         ; d4 := lh_Head
Ø9E2 FCØ9E2     movea.l d4,a1           ; a1 -> nächster Node
Ø9E4            move.l  (a1),d4         ; d4 := ln_Succ
Ø9E6            beq.s   FCØ9EC          ; letzter Node: ->
Ø9E8            addq.l  #4,dØ           ; 4 Bytes für Adresse addieren
Ø9EA            bra.s   FCØ9E2          ; ---> Loop
Ø9EC
Ø9EC FCØ9EC     move.l  #1ØØØ1,d1       ; d1 := Speichertyp: public, clear
Ø9F2            jsr     -C6(a6)         ; ---> AllocMem (für Resident-Tabelle)
Ø9F6            movea.l dØ,a2           ; a2 := dØ -> Speicherbereich
Ø9F8            move.l  dØ,d3           ; d3 := dØ
Ø9FA            move.l  (a3),d4         ; d4 := lh_Head
```

```
Ø9FC FCØ9FC  movea.l d4,a1          ; a1 -> nächster Node
Ø9FE         move.l  (a1),d4        ; d4 := ln_Succ
ØAØØ         beq.s   FCØAØE         ; letzter Node: ->
ØAØ2         move.l  E(a1),(a2)+    ; Zeiger auf Modul in Tabelle eintragen
ØAØ6         moveq   #12,dØ         ; Länge des Node = 18 Bytes
ØAØ8         jsr     -D2(a6)        ; ---> FreeMem
ØAØC         bra.s   FCØ9FC         ; ---> Loop
ØAØE
ØAØE FCØAØE  clr.l   (a2)           ; Endemarke an Tabelle anfügen
ØA1Ø         move.l  d3,dØ          ; dØ := d3 -> Resident-Tabelle
ØA12         rts
ØA14
ØA14 ;----------------------------------
ØA14
ØA14 FCØA14  movem.l a2/a5,-(a7)    ; Register retten
ØA18         move.l  226(a6),dØ     ; dØ := KickTagPtr
ØA1C         beq.s   FCØA36         ; = Ø: ->
ØA1E         movea.l dØ,a2          ; a2 := dØ = KickTagPtr
ØA2Ø FCØA2Ø  move.l  (a2)+,dØ       ; Nächste Adresse aus Liste
ØA22         beq.s   FCØA36         ; Endemarke: ->
ØA24         bmi.s   FCØA2E         ; Link zur Fortsetzungstabelle: ->
ØA26         movea.l dØ,a5          ; a5 := dØ = Adresse aus Liste
ØA28         bsr     FCØ984         ; ---> Node in Liste einfügen
ØA2C         bra.s   FCØA2Ø         ; ---> Loop
ØA2E
ØA2E FCØA2E  bclr.l  #1F,dØ         ; Bit 31 löschen
ØA32         movea.l dØ,a2          ; a2 -> Fortsetzungstabelle
ØA34         bra.s   FCØA2Ø         ; ---> Loop
ØA36
ØA36 FCØA36  movem.l (a7)+,a2/a5    ; Register wiederherstellen
ØA3A         rts
ØA3C
ØA3C ;---------------------------------   SumKickData
ØA3C
ØA3C FCØA3C  movem.l d2-d4,-(a7)    ; Register retten
ØA4Ø         lea     222(a6),aØ     ; aØ -> KickMemPtr
ØA44         movem.l (aØ),d3-d4     ; d3 := KickMemPtr, d4 := KickTagPtr
ØA48         clr.l   (aØ)+          ; KickMemPtr := Ø
ØA4A         clr.l   (aØ)+          ; KickTagPtr := Ø
ØA4C         moveq   #FF,dØ         ; dØ := -1
```

```
ØA4E        move.l  d3,d2           ; d2 := d3 = KickMemPtr
ØA5Ø FCØA5Ø tst.l   d2              ; d2 = Ø?
ØA52        beq.s   FCØA68          ; ja: ->
ØA54        movea.l d2,aØ           ; aØ := d2 = KickMemPtr
ØA56        move.l  (aØ),d2         ; d2 -> nächster Node
ØA58        move.w  E(aØ),d1        ; d1 := ml_NumEntries
ØA5C        add.w   d1,d1           ; mal 2
ØA5E        addi.w  #4,d1           ; + 4 = Länge der ML-Struktur in LW
ØA62        bsr     FCØA8E          ; ---> Prüfsumme in dØ berechnen
ØA66        bra.s   FCØA5Ø          ; ---> Loop
ØA68
ØA68 FCØA68 move.l  d4,d2           ; d2 := d4 = KickTagPtr
ØA6A        beq.s   FCØA8Ø          ; = Ø: ->
ØA6C        movea.l d2,aØ           ; aØ := d2 = KickTagPtr
ØA6E        bra.s   FCØA72          ; --->
ØA7Ø
ØA7Ø FCØA7Ø add.l   d2,dØ           ; d2 zur Prüfsumme addieren
ØA72 FCØA72 move.l  (aØ)+,d2        ; d2 := nächste Adresse aus Tabelle
ØA74        beq.s   FCØA8Ø          ; Endemarke: ->
ØA76        bpl.s   FCØA7Ø          ; Gültige Adresse: ->
ØA78        bclr.l  #1F,d2          ; Bit 31 löschen
ØA7C        movea.l d2,aØ           ; aØ -> Fortsetzungstabelle
ØA7E        bra.s   FCØA72          ; ---> Loop
ØA8Ø
ØA8Ø FCØA8Ø movem.l d3-d4,222(a6)   ; KickMemPtr und KickTagPtr wiederherst.
ØA86        movem.l (a7)+,d2-d4     ; Register wiederherstellen
ØA8A        rts
ØA8C
ØA8C FCØA8C add.l   (aØ)+,dØ        ; nächstes Langwort zu dØ addieren
ØA8E FCØA8E dbra    d1,FCØA8C       ; wiederholen bis d1 = Ø ->
ØA92        rts
ØA94
ØA94 ;------ KickMem-Speicherzuordnung
ØA94
ØA94 FCØA94 move.l  222(a6),d4      ; d4 := KickMemPtr
ØA98 FCØA98 tst.l   d4              ; = Ø?
ØA9A        beq.s   FCØABC          ; ja: ->
ØA9C        movea.l d4,a2           ; a2 := d4 -> ML-Struktur
ØA9E        move.l  (a2),d4         ; d4 -> nächster Node
ØAAØ        lea     E(a2),a2        ; a2 -> ml_NumEntries
```

```
0AA4          move.w  (a2)+,d3           ; d3 := ml_NumEntries
0AA6          moveq   #1,d0              ; d0 := 1 (ok-Flag)
0AA8          bra.s   FC0AB4             ; --->
0AAA
0AAA FC0AAA   movea.l (a2)+,a1           ; a1 := me_Addr
0AAC          move.l  (a2)+,d0           ; d0 := me_Length
0AAE          jsr     -CC(a6)            ; ---> AllocAbs
0AB2          tst.l   d0                 ; Speicher zugeordnet?
0AB4 FC0AB4   dbeq    d3,FC0AAA          ; wenn ja, nächsten Eintrag bearbeiten
0AB8          beq.s   FC0ABE             ; sonst Rückkehr mit d0 = 0
0ABA          bra.s   FC0A98             ; ---> Loop
0ABC
0ABC FC0ABC   moveq   #1,d0              ; d0 := 1 (ok-Flag)
0ABE FC0ABE   rts
0AC0
0AC0 ;--------------------------------   FindResident
0AC0
0AC0 FC0AC0   movem.l a2-a3,-(a7)        ; Register retten
0AC4          movea.l 12C(a6),a2         ; a2 -> Resident-Tabelle
0AC8          movea.l a1,a3              ; a3 := a1 -> Name des Moduls
0ACA FC0ACA   move.l  (a2)+,d0           ; d0 := Modul-Adresse aus Tabelle
0ACC          beq.s   FC0AEA             ; Tabellenende erreicht: ->
0ACE          bgt.s   FC0AD8             ; gültige Adresse: ->
0AD0          bclr.l  #1F,d0             ; Bit 31 löschen ergibt Linkadresse
0AD4          movea.l d0,a2              ; a2 := d0 -> Fortsetzung der Tabelle
0AD6          bra.s   FC0ACA             ; ---> Loop
0AD8
0AD8 FC0AD8   movea.l d0,a1              ; a1 := d0 = Modul-Adresse
0ADA          movea.l a3,a0              ; a3 := a0 -> Name des gesuchten Moduls
0ADC          movea.l E(a1),a1           ; a1 -> Name des Moduls in Tabelle
0AE0 FC0AE0   cmpm.b  (a0)+,(a1)+        ; Namen vergleichen
0AE2          bne.s   FC0ACA             ; bei Ungleichheit: Suche fortsetzen ->
0AE4          tst.b   -1(a0)             ; Endemarke erreicht?
0AE8          bne.s   FC0AE0             ; nein: Vergleich fortsetzen ->
0AEA FC0AEA   movem.l (a7)+,a2-a3        ; Register wiederherstellen
0AEE          rts
0AF0
0AF0 ;--------------------------------   InitCode
0AF0
0AF0 FC0AF0   movem.l d2-d3/a2,-(a7)     ; Register retten
```

```
ØAF4        movea.l 12C(a6),a2         ; a2 -> Resident-Tabelle
ØAF8        move.b  dØ,d2              ; d2 := dØ = rt-Flags
ØAFA        move.b  d1,d3              ; d3 := d1 = Version
ØAFC FCØAFC move.l  (a2)+,dØ           ; dØ := Modul-Adresse aus Tabelle
ØAFE        beq.s   FCØB22             ; Tabellenende erreicht: ->
ØBØØ        bgt.s   FCØBØA             ; gültige Adresse: ->
ØBØ2        bclr.l  #1F,dØ             ; Bit 31 löschen ergibt Linkadresse
ØBØ6        movea.l dØ,a2              ; a2 := dØ -> Fortsetzung der Tabelle
ØBØ8        bra.s   FCØAFC             ; ---> Loop
ØBØA
ØBØA FCØBØA movea.l dØ,a1              ; a1 := Modul-Adresse
ØBØC        cmp.b   B(a1),d3           ; Version kleiner als angefordert?
ØB1Ø        bgt.s   FCØAFC             ; ja: weitersuchen ->
ØB12        move.b  A(a1),dØ           ; dØ := rt_Flags
ØB16        and.b   d2,dØ              ; angegebene Flags gesetzt?
ØB18        beq.s   FCØAFC             ; nein: weitersuchen ->
ØB1A        moveq   #Ø,d1              ; d1 := Ø
ØB1C        jsr     -66(a6)            ; ---> InitResident
ØB2Ø        bra.s   FCØAFC             ; ---> Loop
ØB22
ØB22 FCØB22 movem.l (a7)+,d2-d3/a2     ; Register wiederherstellen
ØB26        rts
ØB28
ØB28 ;--------------------------------- InitResident
ØB28
ØB28 FCØB28 btst.b  #7,A(a1)           ; AutoInit-Flag gesetzt?
ØB2E        bne.s   FCØB3C             ; ja: ->
ØB3Ø        movea.l 16(a1),a1          ; a1 -> Initialisierungs-Routine
ØB34        moveq   #Ø,dØ              ; dØ := Ø
ØB36        movea.l d1,aØ              ; aØ := d1
ØB38        jsr     (a1)               ; ---> Modul initialisieren
ØB3A        bra.s   FCØB7E             ; --->
ØB3C
ØB3C FCØB3C movem.l a1-a2,-(a7)        ; Register retten
ØB4Ø        movea.l 16(a1),a1          ; a1 -> Initialisierungs-Daten
ØB44        movem.l (a1),dØ/aØ-a2      ; Register initialisieren
ØB48        jsr     -54(a6)            ; ---> MakeLibrary
ØB4C        movem.l (a7)+,aØ/a2        ; Register wiederherstellen
ØB5Ø        move.l  dØ,-(a7)           ; Modul-Adresse auf Stack
ØB52        beq.s   FCØB7C             ; nicht genügend Speicher: ->
```

```
0B54         movea.l d0,a1              ; a1 := d0 -> Modul
0B56         move.b  C(a0),d0           ; d0 := rt_Type
0B5A         cmpi.b  #3,d0              ; 'Device'?
0B5E         bne.s   FC0B66             ; nein: ->
0B60         jsr     -1B0(a6)           ; ---> AddDevice
0B64         bra.s   FC0B7C             ; --->
0B66
0B66 FC0B66  cmpi.b  #9,d0              ; 'Library'?
0B6A         bne.s   FC0B72             ; nein: ->
0B6C         jsr     -18C(a6)           ; ---> AddLibrary
0B70         bra.s   FC0B7C             ; --->
0B72
0B72 FC0B72  cmpi.b  #8,d0              ; 'Resource'?
0B76         bne.s   FC0B7C             ; nein: ->
0B78         jsr     -1E6(a6)           ; ---> AddResource
0B7C FC0B7C  move.l  (a7)+,d0           ; d0 := Modul-Adresse vom Stack
0B7E FC0B7E  rts
0B80
0B80 ;--------------------------------- InitStruct Subroutinen
0B80
0B80 ;------ Byte, Repeat
0B80
0B80 FC0B80  move.b  (a1)+,d1           ; nächstes Byte aus Tabelle
0B82 FC0B82  move.b  d1,(a0)+           ; (d0+1)-mal kopieren
0B84         dbra    d0,FC0B82
0B88         bra.s   FC0BD6             ; --->
0B8A
0B8A ;------ APTR, Repeat/Count (nicht zulässig!)
0B8A
0B8A FC0B8A  movem.l d7/a5-a6,-(a7)     ; Register retten
0B8E         move.l  #81000007,d7       ; Alert-Kode 'AN_InitAPtr'
0B94         movea.l 4,a6               ; a6 := SysBase
0B98         jsr     -6C(a6)            ; ---> Alert
0B9C         movem.l (a7)+,d7/a5-a6     ; Register wiederherstellen
0BA0
0BA0 ;------ Langwort, Repeat
0BA0
0BA0 FC0BA0  move.l  a1,d1              ; a1, falls ungerade, auf
0BA2         addq.l  #1,d1              ; nächste Wortgrenze erhöhen
0BA4         andi.b  #FE,d1
```

```
ØBA8        movea.l d1,a1
ØBAA        move.l  (a1)+,d1        ; nächstes Langwort aus Tabelle
ØBAC FCØBAC move.l  d1,(aØ)+        ; (dØ+1)-mal kopieren
ØBAE        dbra    dØ,FCØBAC
ØBB2        bra.s   FCØBD6          ; --->
ØBB4
ØBB4 ;------ Wort, Repeat
ØBB4
ØBB4 FCØBB4 move.l  a1,d1           ; a1, falls ungerade, auf
ØBB6        addq.l  #1,d1           ; nächste Wortgrenze erhöhen
ØBB8        andi.b  #FE,d1
ØBBC        movea.l d1,a1
ØBBE        move.w  (a1)+,d1        ; nächstes Wort aus Tabelle
ØBCØ FCØBCØ move.w  d1,(aØ)+        ; (dØ+1)-mal kopieren
ØBC2        dbra    dØ,FCØBCØ
ØBC6        bra.s   FCØBD6          ; --->
ØBC8
ØBC8 ;----------------------------------  InitStruct
ØBC8
ØBC8 FCØBC8 movea.l a2,aØ           ; aØ := a2 = Adresse der Struktur
ØBCA        lsr.w   #1,dØ           ; dØ := dØ/2 = Länge in Worten
ØBCC        bra.s   FCØBDØ          ; --->
ØBCE
ØBCE FCØBCE clr.w   (aØ)+           ; Speicherwort löschen
ØBDØ FCØBDØ dbra    dØ,FCØBCE       ; falls dØ <> Ø: Löschschleife
ØBD4        movea.l a2,aØ           ; aØ := a2 = Adresse der Struktur
ØBD6 FCØBD6 clr.w   dØ              ; dØ.w löschen
ØBD8        move.b  (a1)+,dØ        ; dØ := Byte aus Tabelle
ØBDA        beq.s   FCØC38          ; Tabellenende erreicht: ->
ØBDC        bclr.l  #7,dØ           ; Bit 7 testen und löschen
ØBEØ        beq.s   FCØBFC          ; Bit 7 war = Ø: ->
ØBE2        bclr.l  #6,dØ           ; Bit 6 testen und löschen
ØBE6        beq.s   FCØBF4          ; Bit 6 war = Ø: ->
ØBE8        subq.l  #1,a1           ; a1 zurücksetzen
ØBEA        move.l  (a1)+,d1        ; d1 := Langwort aus Tabelle
ØBEC        andi.l  #FFFFFF,d1      ; höchstes Byte löschen
ØBF2        bra.s   FCØBF8          ; --->
ØBF4
ØBF4 FCØBF4 moveq   #Ø,d1           ; d1.l löschen
ØBF6        move.b  (a1)+,d1        ; d1 := nächstes Byte aus Tabelle
```

```
ØBF8 FCØBF8  movea.l a2,aØ              ; aØ := a2 = Adresse der Struktur
ØBFA         adda.l  d1,aØ              ; + Zieloffset ergibt Zieladresse in aØ
ØBFC FCØBFC  move.w  dØ,d1              ; d1 := dØ = Kommando-Byte
ØBFE         lsr.w   #3,d1
ØCØØ         andi.w  #E,d1              ; d1 := 'dssØ'
ØCØ4         move.w  34(pc,d1.w),d1     ; d1 := Routinenoffset aus Tabelle
ØCØ8         andi.w  #F,dØ              ; dØ enthält Anzahl 'nnnn'
ØCØC         jmp     2C(pc,d1.w)        ; ---> Routine rufen
ØC1Ø
ØC1Ø ;------ Byte, Count
ØC1Ø
ØC1Ø FCØC1Ø  move.b  (a1)+,(aØ)+        ; Byte aus Tabelle kopieren
ØC12         dbra    dØ,FCØC1Ø          ; (dØ+1)-mal
ØC16         move.l  a1,dØ              ; a1, falls ungerade, auf
ØC18         addq.l  #1,dØ              ; nächste Wortgrenze erhöhen
ØC1A         bclr.l  #Ø,dØ
ØC1E         movea.l dØ,a1
ØC2Ø         bra.s   FCØBD6             ; --->
ØC22
ØC22 ;------ Langwort, Count
ØC22
ØC22 FCØC22  add.w   dØ,dØ              ; Anzahl verdoppeln
ØC24         addq.w  #1,dØ              ; und 1 addieren
ØC26
ØC26 ;------ Wort, Count
ØC26
ØC26 FCØC26  move.l  a1,d1              ; a1, falls ungerade, auf
ØC28         addq.l  #1,d1              ; nächste Wortgrenze erhöhen
ØC2A         andi.b  #FE,d1
ØC2E         movea.l d1,a1
ØC3Ø FCØC3Ø  move.w  (a1)+,(aØ)+        ; Wort kopieren
ØC32         dbra    dØ,FCØC3Ø          ; (dØ+1)-mal
ØC36         bra.s   FCØBD6             ; --->
ØC38
ØC38 FCØC38  rts
ØC3A
ØC3A ;------ Offset-Tabelle der InitStruct-Subroutinen
ØC3A
ØC3A FCØC3A  DC.W    FFE8               ; FCØC22  Langwort, Count
ØC3C         DC.W    FFEC               ; FCØC26  Wort, Count
```

```
ØC3E         DC.W    FFD6               ; FCØC1Ø  Byte, Count
ØC4Ø         DC.W    FF5Ø               ; FCØB8A  APTR, Count
ØC42
ØC42         DC.W    FF66               ; FCØBAØ  Langwort, Repeat
ØC44         DC.W    FF7A               ; FCØBB4  Wort, Repeat
ØC46         DC.W    FF46               ; FCØB8Ø  Byte, Repeat
ØC48         DC.W    FF5Ø               ; FCØB8A  APTR, Repeat
ØC4A         DC.W    Ø
ØC4C
ØC4C ;---------------------------------  Interrupt-Behandlung
ØC4C
ØC4C FCØC4C  movem.l (a7)+,dØ-d1/aØ-a1/a5-a6 ; Register wiederherstellen
ØC5Ø         rte
ØC52
ØC52 ;------ Interrupt Ebene 1
ØC52
ØC52 FCØC52  movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØC56         lea     DFFØØØ,aØ          ; aØ := Spezialchip-Basisadresse
ØC5C         movea.l 4,a6               ; a6 := SysBase
ØC6Ø         move.w  1C(aØ),d1          ; d1 := Interrupt Enable Bits
ØC64         btst.l  #E,d1              ; Interrupts freigegeben?
ØC68         beq.s   FCØC4C             ; nein: fertig ->
ØC6A         and.w   1E(aØ),d1          ; Interrupt Request Bits auswählen
ØC6E         btst.l  #Ø,d1              ; Ser.Port Sendepuffer leer?
ØC72         beq.s   FCØC8Ø             ; nein: ->
ØC74         movem.l 54(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØC7A         pea     -24(a6)            ; Rückkehr nach ExitIntr
ØC7E         jmp     (a5)               ; ---> Interrupt-Routine
ØC8Ø
ØC8Ø FCØC8Ø  btst.l  #1,d1              ; Disk Block beendet?
ØC84         beq.s   FCØC92             ; nein: ->
ØC86         movem.l 6Ø(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØC8C         pea     -24(a6)            ; Rückkehr nach ExitIntr
ØC9Ø         jmp     (a5)               ; ---> Interrupt-Routine
ØC92
ØC92 FCØC92  btst.l  #2,d1              ; Software-Interrupt?
ØC96         beq.s   FCØCA4             ; nein: ->
ØC98         movem.l 6C(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØC9E         pea     -24(a6)            ; Rückkehr nach ExitIntr
ØCA2         jmp     (a5)               ; ---> Interrupt-Routine
```

```
ØCA4
ØCA4 FCØCA4  bra.s    FCØC4C              ; ---> Ausgang
ØCA6
ØCA6 ;------ Interrupt Ebene 2
ØCA6
ØCA6 FCØCA6  movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØCAA         lea      DFFØØØ,aØ           ; aØ := Spezialchip-Basisadresse
ØCBØ         movea.l 4,a6                 ; a6 := SysBase
ØCB4         move.w   1C(aØ),d1           ; d1 := Interrupt Enable Bits
ØCB8         btst.l   #E,d1               ; Interrupts freigegeben?
ØCBC         beq.s    FCØC4C              ; nein: fertig ->
ØCBE         and.w    1E(aØ),d1           ; Interrupt Request Bits auswählen
ØCC2         btst.l   #3,d1               ; Interrupt von IO-Port oder Timer?
ØCC6         beq.s    FCØCD4              ; nein: ->
ØCC8         movem.l 78(a6),a1/a5         ; a1 -> iv_Data, a5 -> iv_Code
ØCCE         pea      -24(a6)             ; Rückkehr nach ExitIntr
ØCD2         jmp      (a5)                ; ---> Interrupt-Routine
ØCD4
ØCD4 FCØCD4  bra      FCØC4C              ; ---> Ausgang
ØCD8
ØCD8 ;------ Interrupt Ebene 3
ØCD8
ØCD8 FCØCD8  movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØCDC         lea      DFFØØØ,aØ           ; aØ := Spezialchip-Basisadresse
ØCE2         movea.l 4,a6                 ; a6 := SysBase
ØCE6         move.w   1C(aØ),d1           ; d1 := Interrupt Enable Bits
ØCEA         btst.l   #E,d1               ; Interrupts freigegeben?
ØCEE         beq      FCØC4C              ; nein: fertig ->
ØCF2         and.w    1E(aØ),d1           ; Interrupt Request Bits auswählen
ØCF6         btst.l   #6,d1               ; Blitter fertig?
ØCFA         beq.s    FCØDØ8              ; nein: ->
ØCFC         movem.l 9C(a6),a1/a5         ; a1 -> iv_Data, a5 -> iv_Code
ØDØ2         pea      -24(a6)             ; Rückkehr nach ExitIntr
ØDØ6         jmp      (a5)                ; ---> Interrupt-Routine
ØDØ8
ØDØ8 FCØDØ8  btst.l   #5,d1               ; Anfang der Bildwechsel-Lücke?
ØDØC         beq.s    FCØD1A              ; nein: ->
ØDØE         movem.l 9Ø(a6),a1/a5         ; a1 -> iv_Data, a5 -> iv_Code
ØD14         pea      -24(a6)             ; Rückkehr nach ExitIntr
ØD18         jmp      (a5)                ; ---> Interrupt-Routine
```

```
ØD1A
ØD1A FCØD1A  btst.l  #4,d1               ; Copper-Interrupt?
ØD1E         beq.s   FCØD2C              ; nein: ->
ØD2Ø         movem.l 84(a6),a1/a5        ; a1 -> iv_Data, a5 -> iv_Code
ØD26         pea     -24(a6)             ; Rückkehr nach ExitIntr
ØD2A         jmp     (a5)                ; ---> Interrupt-Routine
ØD2C
ØD2C FCØD2C  bra     FCØC4C              ; ---> Ausgang
ØD3Ø
ØD3Ø ;------ Interrupt Ebene 4
ØD3Ø
ØD3Ø FCØD3Ø  movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØD34         lea     DFFØØØ,aØ           ; aØ := Spezialchip-Anfangsadresse
ØD3A         movea.l 4,a6                ; a6 := SysBase
ØD3E         move.w  1C(aØ),d1           ; d1 := Interrupt Enable Bits
ØD42         btst.l  #E,d1               ; Interrupts freigegeben?
ØD46         beq     FCØC4C              ; nein: fertig ->
ØD4A         and.w   1E(aØ),d1           ; Interrupt Request Bits auswählen
ØD4E FCØD4E  btst.l  #8,d1               ; Audio-Kanal 1 Block fertig?
ØD52         beq.s   FCØD62              ; nein: ->
ØD54         movem.l B4(a6),a1/a5        ; a1 -> iv_Data, a5 -> iv_Code
ØD5A         pea     FCØDA2              ; Rückkehradresse auf Stack
ØD6Ø         jmp     (a5)                ; ---> Interrupt-Routine
ØD62
ØD62 FCØD62  btst.l  #A,d1               ; Audio-Kanal 3 Block fertig?
ØD66         beq.s   FCØD76              ; nein: ->
ØD68         movem.l CC(a6),a1/a5        ; a1 -> iv_Data, a5 -> iv_Code
ØD6E         pea     FCØDA2              ; Rückkehradresse auf Stack
ØD74         jmp     (a5)                ; ---> Interrupt-Routine
ØD76
ØD76 FCØD76  btst.l  #7,d1               ; Audio-Kanal Ø Block fertig?
ØD7A         beq.s   FCØD8A              ; nein: ->
ØD7C         movem.l A8(a6),a1/a5        ; a1 -> iv_Data, a5 -> iv_Code
ØD82         pea     FCØDA2              ; Rückkehradresse auf Stack
ØD88         jmp     (a5)                ; ---> Interrupt-Routine
ØD8A
ØD8A FCØD8A  btst.l  #9,d1               ; Audio-Kanal 2 Block fertig?
ØD8E         beq.s   FCØD9E              ; nein: ->
ØD9Ø         movem.l CØ(a6),a1/a5        ; a1 -> iv_Data, a5 -> iv_Code
ØD96         pea     FCØDA2              ; Rückkehradresse auf Stack
```

```
ØD9C        jmp      (a5)               ; ---> Interrupt-Routine
ØD9E
ØD9E FCØD9E  bra      FCØC4C             ; ---> Ausgang
ØDA2
ØDA2 FCØDA2  lea      DFFØØØ,aØ          ; aØ := Spezialchip-Basisadresse
ØDA8        movea.l 4,a6               ; a6 := SysBase
ØDAC        move.w  #78Ø,d1            ; liegt noch ein Audio-Interrupt vor?
ØDBØ        and.w   1C(aØ),d1
ØDB4        and.w   1E(aØ),d1
ØDB8        bne.s   FCØD4E             ; ja: ->
ØDBA        jmp     -24(a6)            ; ---> ExitIntr
ØDBE
ØDBE ;------ Interrupt Ebene 5
ØDBE
ØDBE FCØDBE  movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØDC2        lea      DFFØØØ,aØ          ; aØ := Spezialchip-Basisadresse
ØDC8        movea.l 4,a6               ; a6 := SysBase
ØDCC        move.w  1C(aØ),d1          ; d1 := Interrupt Enable Bits
ØDDØ        btst.l  #E,d1              ; Interrupts freigegeben?
ØDD4        beq     FCØC4C             ; nein: fertig ->
ØDD8        and.w   1E(aØ),d1          ; Interrupt Request Bits auswählen
ØDDC        btst.l  #C,d1              ; Disk-Synchronisation?
ØDEØ        beq.s   FCØDEE             ; nein: ->
ØDE2        movem.l E4(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØDE8        pea     -24(a6)            ; Rückkehr nach ExitIntr
ØDEC        jmp     (a5)               ; ---> Interrupt-Routine
ØDEE
ØDEE FCØDEE  btst.l  #B,d1              ; Ser. Port Empfangspuffer voll?
ØDF2        beq.s   FCØEØØ             ; nein: ->
ØDF4        movem.l D8(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØDFA        pea     -24(a6)            ; Rückkehr nach ExitIntr
ØDFE        jmp     (a5)               ; ---> Interrupt-Routine
ØEØØ
ØEØØ FCØEØØ  bra      FCØC4C             ; ---> Ausgang
ØEØ4
ØEØ4 ;------ Interrupt Ebene 6
ØEØ4
ØEØ4 FCØEØ4  movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØEØ8        lea      DFFØØØ,aØ          ; aØ := Spezialchip-Basisadresse
ØEØE        movea.l 4,a6               ; a6 := SysBase
```

```
ØE12        move.w  1C(aØ),d1          ; d1 := Interrupt Enable Bits
ØE16        btst.l  #E,d1              ; Interrupts freigegeben?
ØE1A        beq     FCØC4C             ; nein: fertig ->
ØE1E        and.w   1E(aØ),d1          ; Interrupt Request Bits auswählen
ØE22        btst.l  #E,d1              ; Master Interrupt?
ØE26        beq.s   FCØE34             ; nein: ->
ØE28        movem.l FC(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØE2E        pea     -24(a6)            ; Rückkehr nach ExitIntr
ØE32        jmp     (a5)               ; ---> Interrupt-Routine
ØE34
ØE34 FCØE34 btst.l  #D,d1              ; Externer Interrupt?
ØE38        beq.s   FCØE46             ; nein: ->
ØE3A        movem.l FØ(a6),a1/a5       ; a1 -> iv_Data, a5 -> iv_Code
ØE4Ø        pea     -24(a6)            ; Rückkehr nach ExitIntr
ØE44        jmp     (a5)               ; ---> Interrupt-Routine
ØE46
ØE46 FCØE46 bra     FCØC4C             ; ---> Ausgang
ØE4A
ØE4A ;------ Interrupt Ebene 7
ØE4A
ØE4A FCØE4A movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØE4E        movea.l 4,a6               ; a6 := SysBase
ØE52        movem.l 1Ø8(a6),a1/a5      ; a1 -> iv_Data, a5 -> iv_Code
ØE58        jsr     (a5)               ; ---> Interrupt-Routine
ØE5A        movem.l (a7)+,dØ-d1/aØ-a1/a5-a6 ; Register wiederherstellen
ØE5E        rte
ØE6Ø
ØE6Ø ;---------------------------------   ExitIntr
ØE6Ø
ØE6Ø FCØE6Ø btst.b  #5,18(a7)          ; Interrupt aus Supervisor-Modus?
ØE66        bne.s   FCØE8Ø             ; ja: ->
ØE68        movea.l 4,a6               ; a6 := SysBase
ØE6C        tst.b   127(a6)            ; Forbid gesetzt?
ØE7Ø        bge.s   FCØE8Ø             ; ja: ->
ØE72        btst.b  #7,124(a6)         ; Scheduling Attention Flag gesetzt?
ØE78        beq.s   FCØE8Ø             ; nein: ->
ØE7A        move    #2ØØØ,sr           ; IR-Ebene Ø setzen
ØE7E        bra.s   FCØE8A             ; ---> Schedule
ØE8Ø
ØE8Ø FCØE8Ø movem.l (a7)+,dØ-d1/aØ-a1/a5-a6 ; Register wiederherstellen
```

```
ØE84        rte
ØE86
ØE86 ;--------------------------------   Schedule
ØE86
ØE86 FCØE86 movem.l dØ-d1/aØ-a1/a5-a6,-(a7) ; Register retten
ØE8A FCØE8A move    #27ØØ,sr           ; IR-Ebene 7 setzen
ØE8E        bclr.b  #7,124(a6)         ; Scheduling Attention Flag löschen
ØE94        movea.l 114(a6),a1         ; a1 -> ThisTask
ØE98        btst.b  #5,E(a1)           ; Exception-Flag gesetzt?
ØE9E        bne.s   FCØEBE             ; ja: ->
ØEAØ        lea     196(a6),aØ         ; aØ -> Ready List Header
ØEA4        cmpa.l  8(aØ),aØ           ; Liste leer?
ØEA8        beq.s   FCØE8Ø             ; ja: ->
ØEAA        movea.l (aØ),aØ            ; aØ -> 1. Task in der Liste
ØEAC        move.b  9(aØ),d1           ; d1 := Priorität
ØEBØ        cmp.b   9(a1),d1           ; >= Priorität der laufenden Task?
ØEB4        bge.s   FCØEBE             ; ja: ->
ØEB6        btst.b  #6,124(a6)         ; Zeitquantum abgelaufen?
ØEBC        beq.s   FCØE8Ø             ; nein: ->
ØEBE FCØEBE lea     196(a6),aØ         ; aØ -> Ready List Header
ØEC2        bsr     FC1634             ; ---> Enqueue laufende Task
ØEC6        move.b  #3,F(a1)           ; tc_State := 'Ready'
ØECC        move    #2ØØØ,sr           ; IR-Ebene := Ø
ØEDØ        movem.l (a7)+,dØ-d1/aØ-a1/a5 ; Register wiederherstellen
ØED4        move.l  (a7),-(a7)         ; a6 im Stack nach unten kopieren
ØED6        move.l  -34(a6),4(a7)      ; Adresse von 'Switch' darübersetzen
ØEDC        movea.l (a7)+,a6           ; a6 vom Stack wiederherstellen
ØEDE        rts
ØEEØ
ØEEØ ;--------------------------------   Switch
ØEEØ
ØEEØ FCØEEØ move    #2ØØØ,sr           ; IR-Ebene := Ø
ØEE4        move.l  a5,-(a7)           ; a5 retten
ØEE6        move    usp,a5             ; a5 := Rücksprungadresse aus User-Stack
ØEE8        movem.l dØ-d7/aØ-a6,-(a5)  ; Register auf User-Stack retten
ØEEC        movea.l 4,a6               ; a6 := SysBase
ØEFØ        move.w  126(a6),dØ         ; dØ := Forbid/Disable-Ebene
ØEF4        move.w  #FFFF,126(a6)      ; Forbid/Disable rücksetzen
ØEFA        move.w  #CØØØ,INTENA       ; Interrupts freigeben
ØFØ2        move.l  (a7)+,34(a5)       ; a5 im User-Stack wiederherstellen
```

```
ØFØ6          move.w  (a7)+,-(a5)         ; Status-Register auf User-Stack
ØFØ8          move.l  (a7)+,-(a5)         ; Rückkehradresse auf User-Stack
ØFØA          lea     9A(pc),a4           ; a4 := FCØFA6
ØFØE FCØFØE   movea.l 114(a6),a3          ; a3 -> ThisTask
ØF12          move.w  dØ,1Ø(a3)           ; Forbid/Disable-Ebene eintragen
ØF16          move.l  a5,36(a3)           ; tc-SPReg := User Stackpointer
ØF1A          btst.b  #6,E(a3)            ; Switch-Flag in tc_Flags gesetzt?
ØF2Ø          beq.s   FCØF3C              ; nein: ->
ØF22          movea.l 42(a3),a5           ; a5 -> tc_Switch
ØF26          jsr     (a5)                ; ---> Switch-Routine
ØF28          bra.s   FCØF3C              ; ---> Dispatch
ØF2A
ØF2A ;---------------------------------   Dispatch
ØF2A
ØF2A FCØF2A   lea     7A(pc),a4           ; a4 := FCØFA6
ØF2E FCØF2E   move.w  #FFFF,126(a6)       ; Forbid/Disable rücksetzen
ØF34          move.w  #CØØØ,INTENA        ; Interrupts freigeben
ØF3C FCØF3C   lea     196(a6),aØ          ; aØ -> Ready List Header
ØF4Ø FCØF4Ø   move    #27ØØ,sr            ; IR-Ebene := 7
ØF44          movea.l (aØ),a3             ; a3 -> 1. Task in Ready List
ØF46          move.l  (a3),dØ             ; dØ := ln_Succ
ØF48          bne.s   FCØF5A              ; Liste nicht leer: ->
ØF4A          addq.l  #1,118(a6)          ; IdleCount inkrementieren
ØF4E          bset.b  #7,124(a6)          ; Scheduling Attention Flag setzen
ØF54          stop    #2ØØØ               ; STOP bis Int
ØF58          bra.s   FCØF4Ø              ; ---> Loop
ØF5A
ØF5A FCØF5A   move.l  dØ,(aØ)             ; Node aus Ready List entfernen
ØF5C          movea.l dØ,a1
ØF5E          move.l  aØ,4(a1)
ØF62          addq.l  #1,11C(a6)          ; DispCount inkrementieren
ØF66          move.l  a3,114(a6)          ; ThisTask := a3
ØF6A          move.w  12Ø(a6),122(a6)     ; Zeitquantum in Zähler kopieren
ØF7Ø          bclr.b  #6,124(a6)          ; Flag für Zeitablauf löschen
ØF76          move.b  #2,F(a3)            ; tc_State := 'Run'
ØF7C          move.w  1Ø(a3),126(a6)      ; Forbid/Disable-Ebene eintragen
ØF82          tst.b   126(a6)             ; Disable gesetzt?
ØF86          bmi.s   FCØF9Ø              ; nein: ->
ØF88          move.w  #4ØØØ,INTENA        ; Interrupts abschalten
ØF9Ø FCØF9Ø   move    #2ØØØ,sr            ; IR-Ebene := Ø
```

```
0F94         move.b  E(a3),d0            ; d0 := tc_Flags
0F98         andi.b  #A0,d0              ; Except oder Launch gesetzt?
0F9C         beq.s   FC0FA0              ; nein: ->
0F9E         bsr.s   FC0FB6              ; ---> Launch/Exception-Routine
0FA0 FC0FA0  movea.l 36(a3),a5           ; a5 := tc_SPReg = Task-Stackpointer
0FA4         jmp     (a4)                ; --->
0FA6
0FA6 FC0FA6  lea     42(a5),a2           ; a2 -> Rückkehradresse vom User-Stack
0FAA         move    a2,usp              ; User-Stackpointer setzen
0FAC         move.l  (a5)+,-(a7)         ; PC bei Interrupt auf System-Stack
0FAE         move.w  (a5)+,-(a7)         ; SR bei Interrupt auf System-Stack
0FB0         movem.l (a5),d0-d7/a0-a6    ; Register wiederherstellen
0FB4         rte                         ; ---> Task starten
0FB6
0FB6 FC0FB6  btst.l  #7,d0               ; 'Launch'?
0FBA         beq.s   FC0FC6              ; nein: ->
0FBC         move.b  d0,d2               ; d0 in d2 retten
0FBE         movea.l 46(a3),a5           ; a5 -> tc_Launch
0FC2         jsr     (a5)                ; ---> Launch-Routine
0FC4         move.b  d2,d0               ; d0 wiederherstellen
0FC6 FC0FC6  btst.l  #5,d0               ; 'Except'?
0FCA         bne.s   FC0FCE              ; ja: ->
0FCC         rts
0FCE
0FCE ;---------------------------------  Exception
0FCE
0FCE FC0FCE  bclr.b  #5,E(a3)            ; Except-Flag rücksetzen
0FD4         move.w  #4000,INTENA        ; Interrupts abschalten
0FDC         addq.b  #1,126(a6)          ; Disable-Ebene inkrementieren
0FE0         move.l  1A(a3),d0           ; d0 := tc_SigRecvd
0FE4         and.l   1E(a3),d0           ; tc_SigExcept auswählen
0FE8         eor.l   d0,1E(a3)           ; Empfangene Except-Signale löschen
0FEC         eor.l   d0,1A(a3)
0FF0         subq.b  #1,126(a6)          ; Disable-Ebene dekrementieren
0FF4         bge.s   FC0FFE              ; Interrupts noch gesperrt: ->
0FF6         move.w  #C000,INTENA        ; Interrupts freigeben
0FFE FC0FFE  movea.l 36(a3),a1           ; a1 := tc_SPReg
1002         move.l  E(a3),-(a1)         ; tc_Flags auf Stack
1006         tst.b   126(a6)             ; Disable-Ebene
100A         bne.s   FC101A              ; nicht Null: ->
```

```
100C         subq.b  #1,126(a6)        ; dekrementieren
1010         bge.s   FC101A            ; Interrupts noch gesperrt: ->
1012         move.w  #C000,INTENA      ; Interrupts freigeben
101A FC101A  move.l  #FC103A,-(a1)     ; Adresse auf Stack
1020         move    a1,usp            ; User-Stackpointer setzen
1022         btst.b  #0,129(a6)        ; Prozessor M 68010+ ?
1028         beq.s   FC102E            ; nein: ->
102A         move.w  #20,-(a7)         ; Formatkode, Vektoradresse auf Stack
102E FC102E  move.l  2A(a3),-(a7)      ; *tc_ExceptCode auf Stack
1032         clr.w   -(a7)             ; Statusregister auf Stack := 0
1034         movea.l 26(a3),a1         ; a1 -> tc_ExceptData
1038         rte                       ; ---> ExceptCode
103A
103A FC103A  movea.l 4,a6              ; a6 := SysBase
103E         lea     6(pc),a5          ; a5 := Rückkehradresse FC1046
1042         jmp     -1E(a6)           ; ---> Supervisor
1046
1046 FC1046  lea     -A2(pc),a4        ; a4 := FC0FA6
104A         btst.b  #0,129(a6)        ; Prozessor M 68010+ ?
1050         beq.s   FC1060            ; nein: ->
1052         addq.l  #2,a7             ; Stackpointer um 2 erhöhen
1054         btst.b  #4,129(a6)        ; 68881 im System?
105A         beq.s   FC1060            ; nein: ->
105C         lea     92(pc),a4         ; a4 := FC10F0
1060 FC1060  addq.l  #6,a7             ; RTE-Daten vom Stack nehmen
1062         movea.l 114(a6),a3        ; a3 -> ThisTask
1066         or.l    d0,1E(a3)         ; Except-Signale wieder setzen
106A         move    usp,a1            ; a1 := User-Stackpointer
106C         move.l  (a1)+,E(a3)       ; tc_Flags zurückspeichern
1070         move.l  a1,36(a3)         ; tc_SPReg zurückspeichern
1074         move.w  10(a3),126(a6)    ; Forbid/Disable-Ebene wiederherstellen
107A         tst.b   126(a6)           ; Disable gesetzt?
107E         bmi.s   FC1088            ; nein: ->
1080         move.w  #4000,INTENA      ; Interrupts abschalten
1088 FC1088  rts
108A
108A ;-----  Switch für MC 68881
108A
108A FC108A  move    #2000,sr          ; IR-Ebene := 0
108E         move.l  a5,-(a7)          ; a5 auf Supervisor-Stack retten
```

```
1090         move     usp,a5            ; a5 := User Stackpointer
1092         movem.l  d0-d7/a0-a6,-(a5) ; Register auf User Stack retten
1096         movea.l  4,a6              ; a6 := SysBase
109A         move.w   126(a6),d0        ; d0 := IDNestCnt
109E         move.w   #FFFF,126(a6)     ; Enable/Permit rücksetzen
10A4         move.w   #C000,INTENA      ; Interrupts freigeben
10AC         move.l   (a7)+,34(a5)      ; a5 im Stack wiederherstellen
10B0         move.w   (a7)+,-(a5)       ; SSR in User Stack übertragen
10B2         move.l   (a7)+,-(a5)       ; PC in User Stack übertragen
10B4         cpsave   -(a5)             ; Coprocessor Status auf Stack retten
10B6         tst.b    (a5)              ; a5 = 0?
10B8         beq.s    FC10E0            ; ja: ->
10BA         moveq    #FF,d2            ; d2 := -1
10BC         move.w   (a7),d1           ; Nächstes Wort vom Systemstack
10BE         andi.w   #F000,d1          ; Formatkode isolieren
10C2         cmpi.w   #9000,d1          ; Formatkode = 9?
10C6         bne.s    FC10D6            ; Nein: ->
10C8         addq.l   #2,a7             ; Systemstackzeiger um 2 erhöhen
10CA         move.l   (a7)+,-(a5)       ; Die nächsten drei Langworte
10CC         move.l   (a7)+,-(a5)       ; aus dem Systemstack
10CE         move.l   (a7)+,-(a5)       ; in den Userstack übertragen
10D0         move.w   #20,-(a7)
10D4         move.w   d1,d2
10D6 FC10D6  cpgen    #E0FF,-(a5)       ; Koprozessorbefehle generieren
10DA         cpgen    #BC00,-(a5)
10DE         move.w   d2,-(a5)          ; Formatkode auf Userstack legen
10E0 FC10E0  lea      E(pc),a4          ; a4 := FC10F0 (Rückkehradresse)
10E4         bra      FC0F0E            ; ---> Switch-Einsprung
10E8
10E8 ;------ Dispatch für MC 68881
10E8
10E8 FC10E8  lea      6(pc),a4          ; a4 := FC10F0 (Rückkehradresse)
10EC         bra      FC0F2E            ; ---> Dispatch
10F0
10F0 FC10F0  move.b   (a5),d0           ; d0 := Formatkode vom Stack
10F2         beq.s    FC1110            ; Formatkode = 0: ->
10F4         addq.l   #2,a5             ; Formatkode übergehen
10F6         cpgen    (a5)+,#9C00       ; Koprozessorbefehle generieren
10FA         cpgen    (a5)+,#D0FF
10FE         cmpi.b   #90,d0            ; War Formatkode = 9?
```

```
1102         bne.s    FC1110          ; Nein: ->
1104         addq.l   #2,a7           ; Systemstackzeiger um 2 erhöhen
1106         move.l   (a5)+,-(a7)     ; Die nächsten drei Langworte
1108         move.l   (a5)+,-(a7)     ; vom Userstack in den
110A         move.l   (a5)+,-(a7)     ; Systemstack übertragen
110C         move.w   #9020,-(a7)
1110 FC1110  cprestore (a5)+          ; Coprocessor Status wiederherstellen
1112         lea      42(a5),a2       ; User Stackpointer wiederherstellen
1116         move     a2,usp
1118         move.l   (a5)+,-(a7)     ; PC in System-Stack übertragen
111A         move.w   (a5)+,-(a7)     ; SSR in System-Stack übertragen
111C         movem.l (a5),d0-d7/a0-a6 ; Register wiederherstellen
1120         rte
1122
1122 ;--------------------------------- SetSR
1122
1122 FC1122  movea.l a5,a0            ; a5 in a0 retten
1124         lea      8(pc),a5        ; a5 := Rückkehradresse FC112E
1128         jsr      -1E(a6)         ; ---> Supervisor
112C         rts
112E
112E FC112E  movea.l a0,a5            ; a5 wiederherstellen
1130         movea.w (a7),a0          ; a0 := Statusregister
1132         and.w    d1,d0           ; d0 mit d1 maskieren
1134         not.w    d1
1136         and.w    d1,(a7)         ; Statusregister ändern
1138         or.w     d0,(a7)
113A         moveq    #0,d0
113C         move.w   a0,d0           ; d0 := altes Statusregister
113E         rte
1140
1140 ;--------------------------------- GetCC
1140
1140 FC1140  move     sr,d0           ; d0 := Statusregister
1142         andi.w   #FF,d0          ; CCR-Teil isolieren
1146         rts
1148
1148 ;--------------------------------- SuperState
1148
1148 FC1148  movea.l a5,a0            ; a5 in a0 retten
```

```
114A        lea      8(pc),a5          ; a5 := FC1154 (Rückkehradresse)
114E        jsr      -1E(a6)           ; ---> Supervisor
1152        rts
1154
1154 FC1154 movea.l a0,a5              ; a5 wiederherstellen
1156        clr.l    d0                ; d0 := 0
1158        bset.b   #5,(a7)           ; S-Bit im SR im Stack setzen
115C        bne.s    FC1172            ; war bereits gesetzt: ->
115E        move     (a7)+,sr          ; SR vom Stack laden
1160        move.l   a7,d0             ; d0 := SSP
1162        move     usp,a7            ; a7 := USP
1164        btst.b   #0,129(a6)        ; M 68000 Prozessor?
116A        beq.s    FC116E            ; ja: ->
116C        addq.l   #2,d0             ; sonst ein Wort mehr vom Stack nehmen
116E FC116E addq.l   #4,d0             ; Rücksprungadresse entfernen
1170        rts
1172
1172 FC1172 rte
1174
1174 ;--------------------------------- UserState
1174
1174 FC1174 movea.l a5,a0              ; a5 in a0 retten
1176        lea      6(pc),a5          ; a5 := FC117E (Rückkehradresse)
117A        jmp      -1E(a6)           ; ---> Supervisor
117E
117E FC117E movea.l a0,a5              ; a5 wiederherstellen
1180        move.w   (a7)+,d1          ; d1 := SR vom Stack
1182        move     a7,usp            ; USP := SSP
1184        movea.l  d0,a7             ; SSP := d0
1186        bclr.l   #D,d1             ; S-Bit zurücksetzen
118A        move     d1,sr
118C        rts
118E
118E ;--------------------------------- SetIntVector
118E
118E FC118E mulu     #C,d0             ; d0 := Interrupt-Nummer mal 12
1192        lea      54(a6,d0.w),a0    ; a0 -> Interrupt-Vektor
1196        move.w   #4000,INTENA      ; Alle Interrupts sperren
119E        addq.b   #1,126(a6)        ; Disable
11A2        move.l   8(a0),d0          ; d0 -> iv_Node alt
```

```
11A6        move.l  a1,8(aØ)            ; a1 -> iv_Node neu eintragen
11AA        beq.s   FC11BA              ; kein gültiger Zeiger: ->
11AC        move.l  E(a1),Ø(aØ)         ; iv_Data eintragen
11B2        move.l  12(a1),4(aØ)        ; iv_Code eintragen
11B8        bra.s   FC11C4              ; --->
11BA
11BA FC11BA moveq   #FF,d1              ; d1 := -1
11BC        move.l  d1,Ø(aØ)            ; iv_Data := -1
11CØ        move.l  d1,4(aØ)            ; iv_Code := -1
11C4 FC11C4 subq.b  #1,126(a6)          ; Enable
11C8        bge.s   FC11D2              ; Noch keine Freigabe: ->
11CA        move.w  #CØØØ,INTENA        ; Alle Interrupts freigeben
11D2 FC11D2 rts
11D4
11D4 ;---------------------------------  AddIntServer
11D4
11D4 FC11D4 move.l  d2,-(a7)            ; d2 retten
11D6        move.l  dØ,d2               ; d2 := Interrupt-Nummer
11D8        move.l  dØ,d1               ; d1 := Interrupt-Nummer
11DA        mulu    #C,dØ               ; dØ := Interrupt-Nummer mal 12
11DE        lea     54(a6,dØ.w),aØ      ; aØ -> Interrupt-Vektor
11E2        movea.l Ø(aØ),aØ            ; aØ := iv_Data
11E6        move.w  #4ØØØ,INTENA        ; alle Interrupts sperren
11EE        addq.b  #1,126(a6)          ; Disable
11F2        bsr     FC1634              ; ---> Enqueue
11F6        move.w  #8ØØØ,dØ            ; Neues Interrupt-Bit setzen
11FA        bset.l  d2,dØ
11FC        move.w  dØ,INTENA
12Ø2        subq.b  #1,126(a6)          ; Enable
12Ø6        bge.s   FC121Ø              ; Noch keine Freigabe: ->
12Ø8        move.w  #CØØØ,INTENA        ; Alle Interrupts freigeben
121Ø FC121Ø move.l  (a7)+,d2            ; d2 wiederherstellen
1212        rts
1214
1214 ;---------------------------------  RemIntServer
1214
1214 FC1214 move.l  d2,-(a7)            ; d2 retten
1216        mulu    #C,dØ               ; dØ := Interrupt-Nummer mal 12
121A        lea     54(a6,dØ.w),aØ      ; aØ -> Interrupt-Vektor
121E        movea.l Ø(aØ),aØ            ; aØ := iv_Data
```

```
1222        move.l  d0,d2               ; d2 := Interrupt-Nummer mal 12
1224        move.l  a0,d1               ; d1 := iv_Data
1226        move.w  #4000,INTENA        ; Alle Interrupts sperren
122E        addq.b  #1,126(a6)          ; Disable
1232        bsr     FC1600              ; ---> Remove
1236        movea.l d1,a0               ; a0 := iv_Data
1238        cmpa.l  8(a0),a0            ; Liste jetzt leer?
123C        bne     FC124A              ; nein: ->
1240        moveq   #0,d1               ; Interrupt Enable Bit rücksetzen
1242        bset.l  d2,d1
1244        move.w  d1,INTENA
124A FC124A subq.b  #1,126(a6)          ; Enable
124E        bge.s   FC1258              ; Noch keine Freigabe: ->
1250        move.w  #C000,INTENA        ; Alle Interrupts freigeben
1258 FC1258 move.l  (a7)+,d2            ; d2 wiederherstellen
125A        rts
125C
125C ;------ Interrupt Server initialisieren
125C
125C FC125C movem.l d2-d3/a2-a3,-(a7)  ; Register retten
1260        move.l  #6E,d0              ; d0 := 110 = benötigte Speicherbytes
1266        move.l  #10001,d1           ; d1 := Speichertyp PUBLIC, CLEAR
126C        bsr     FC1794              ; ---> AllocMem
1270        tst.l   d0                  ; Speicherplatz zugewiesen?
1272        bne.s   FC128A              ; ja: ->
1274        movem.l d7/a5-a6,-(a7)      ; Register retten
1278        move.l  #81000006,d7        ; d7 := Alert-Kode 'AN_IntrMem'
127E        movea.l 4,a6                ; a6 := SysBase
1282        jsr     -6C(a6)             ; ---> Alert
1286        movem.l (a7)+,d7/a5-a6      ; Register wiederherstellen
128A FC128A moveq   #4,d2               ; d2 := 4 (Zähler)
128C        movea.l d0,a2               ; a2 := d0 -> Speicherbereich
128E        lea     4E(pc),a3           ; a3 := FC12DE (Initialisierungstabelle)
1292 FC1292 move.l  a2,d1               ; d1 := a2 -> Speicherbereich
1294        move.l  a2,(a2)             ; List Header am Anfang des
1296        addq.l  #4,(a2)             ; Speicherbereichs erzeugen
1298        clr.l   4(a2)
129C        move.l  a2,8(a2)
12A0        lea     E(a2),a2            ; a2 := a2+14 -> hinter den Header
12A4        move.w  (a3)+,d3            ; d3 := Int-Nummer aus Tabelle
```

```
12A6        moveq   #0,d0              ; d0 := 0
12A8        bset.l  d3,d0              ; Bit Nummer (d3) setzen
12AA        move.w  d0,(a2)+           ; Sperrwort speichern
12AC        bset.l  #F,d0              ; Bit Nummer 15 setzen
12B0        move.w  d0,(a2)+           ; Freigabewort speichern
12B2        move.w  (a3)+,(a2)+        ; Nächstes Wort aus Tabelle kopieren
12B4        move.w  (a3)+,(a2)+        ; Nächstes Wort aus Tabelle kopieren
12B6        lea     44(pc),a1          ; a1 := FC12FC (Int.Service-Routine)
12BA        mulu    #C,d3              ; d3 := Interrupt-Nummer mal 12
12BE        movem.l d1/a1,54(a6,d3.w)  ; iv_Data und iv_Code eintragen
12C4        dbra    d2,FC1292          ; das ganze 5 mal ->
12C8        move.l  #FC1380,70(a6)     ; SoftInt-Codeadresse speichern
12D0        move.w  #8004,INTENA       ; Soft Interrupts freigeben
12D8        movem.l (a7)+,d2-d3/a2-a3  ; Register wiederherstellen
12DC        rts
12DE
12DE ;------ Tabelle für Initialisierung der Interrupt Server
12DE
12DE FC12DE DC.W    3                  ; Int 3: IO-Ports und Timer
12E0        DC.W    8
12E2        DC.W    0
12E4        DC.W    5                  ; Int 5: Bildwechsel-Interrupt
12E6        DC.W    20
12E8        DC.W    0
12EA        DC.W    4                  ; Int 4: Copper
12EC        DC.W    10
12EE        DC.W    0
12F0        DC.W    D                  ; Int 13: Externer Interrupt
12F2        DC.W    2000
12F4        DC.W    0
12F6        DC.W    F                  ; Int 15: NMI
12F8        DC.W    0
12FA        DC.W    0
12FC
12FC ;------ Interrupt-Routine für Int 3, 4, 5, 13, 15
12FC
12FC FC12FC move.w  12(a1),-(a7)       ; Int-Request Maske auf Stack
1300        move.l  a2,-(a7)           ; a2 retten
1302        movea.l (a1),a2            ; a2 -> erster Node
1304 FC1304 move.l  (a2),d0            ; d0 -> ln_Succ
```

```
1306        beq.s   FC1316          ; Liste zu Ende: ->
1308        movem.l E(a2),a1/a5     ; a1 -> iv_Data, a5 -> iv_Code
130E        jsr     (a5)            ; ---> Interrupt-Code
1310        bne.s   FC1316          ; Liste fertig: ->
1312        movea.l (a2),a2         ; sonst: a2 -> nächster Node
1314        bra.s   FC1304          ; ---> Loop
1316
1316 FC1316 movea.l (a7)+,a2       ; a2 wiederherstellen
1318        move.w  (a7)+,INTREQ    ; Int-Request vom Stack setzen
131E        rts
1320
1320 ;-------------------------------- Cause
1320
1320 FC1320 move.w  #4000,INTENA    ; Interrupts sperren
1328        addq.b  #1,126(a6)      ; Disable
132C        cmpi.b  #B,8(a1)        ; Node-Typ 'SoftInt'?
1332        beq.s   FC1370          ; ja: ->
1334        move.b  #B,8(a1)        ; Typ 'SoftInt' in Node eintragen
133A        moveq   #0,d0           ; d0 := 0
133C        move.b  9(a1),d0        ; d0 := Priorität
1340        andi.w  #F0,d0          ; modulo 16
1344        ext.w   d0              ; auf d0.w erweitern
1346        lea     1D2(a6),a0      ; a0 -> 3. List Header für SoftInt
134A        adda.w  d0,a0           ; a0 -> List Header gemäss Priorität
134C        lea     4(a0),a0        ; Neuen Interrupt am Ende der Liste
1350        move.l  4(a0),d0        ; anfügen
1354        move.l  a1,4(a0)
1358        move.l  a0,(a1)
135A        move.l  d0,4(a1)
135E        movea.l d0,a0
1360        move.l  a1,(a0)
1362        bset.b  #5,124(a6)      ; SoftInt Request Flag setzen
1368        move.w  #8004,INTREQ    ; Soft Interrupt anfordern
1370 FC1370 subq.b  #1,126(a6)      ; Enable
1374        bge.s   FC137E          ; Noch keine Freigabe: ->
1376        move.w  #C000,INTENA    ; Interrupts freigeben
137E FC137E rts
1380
1380 ;-------------------------------- SoftInt-Routine
1380
```

```
1380 FC1380  move.w  #4,INTREQ          ; Interrupt-Anforderung löschen
1388         bclr.b  #5,124(a6)         ; SoftInt Request Flag löschen
138E         bne.s   FC1392             ; war gesetzt: ->
1390         rts
1392
1392 FC1392  move.w  #4,INTENA          ; Soft Interrupts sperren
139A         bra.s   FC13C2             ; --->
139C
139C FC139C  move    #2700,sr           ; Int Ebene 7 setzen
13A0         movea.l (a0),a1            ; Ersten Node entfernen (RemHead)
13A2         move.l  (a1),d0
13A4         beq.s   FC13AE
13A6         move.l  d0,(a0)
13A8         exg     d0,a1
13AA         move.l  a0,4(a1)
13AE FC13AE  movea.l d0,a1              ; a1 -> Interrupt Node
13B0         move.b  #2,8(a1)           ; Type 'Interrupt' eintragen
13B6         move    #2000,sr           ; Interrupt Ebene 0 setzen
13BA         movem.l E(a1),a1/a5        ; a1 -> iv_Data, a5 -> iv_Code
13C0         jsr     (a5)               ; ---> Interrupt-Code
13C2 FC13C2  moveq   #4,d0              ; d0 := 4 (Zähler)
13C4         lea     1F2(a6),a0         ; a0 -> SoftInt Header Priorität 32
13C8         move.w  #4,INTREQ          ; Soft Interrupts sperren
13D0 FC13D0  cmpa.l  8(a0),a0           ; Liste leer?
13D4         bne.s   FC139C             ; nein: ->
13D6         lea     -10(a0),a0         ; a0 -> nächster SoftInt List Header
13DA         dbra    d0,FC13D0          ; insgesamt 5 mal ->
13DE         move    #2100,sr           ; Interrupt Ebene 1 setzen
13E2         move.w  #8004,INTENA       ; SoftInt anfordern
13EA         rts
13EC
13EC ;---------------------------------  Disable
13EC
13EC FC13EC  move.w  #4000,INTENA       ; Interrupts sperren
13F4         addq.b  #1,126(a6)         ; Disable-Ebene erhöhen
13F8         rts
13FA
13FA ;---------------------------------  Enable
13FA
13FA FC13FA  subq.b  #1,126(a6)         ; Disable-Ebene erniedrigen
```

```
13FE        bge.s   FC1408           ; Noch keine Freigabe: ->
1400        move.w  #C000,INTENA     ; Interrupts freigeben
1408 FC1408 rts
140A
140A        DC.W    0
140C
140C ;---------------------------------  AddLibrary
140C
140C FC140C lea     17A(a6),a0       ; a0 -> Library List Header
1410        bsr     FC1682           ; ---> Forbid, Enqueue, Permit
1414        bsr     FC1498           ; ---> SumLibrary
1418        rts
141A
141A ;---------------------------------  RemLibrary
141A
141A FC141A move.l  a1,d0            ; d0 := a1 -> LibBase
141C        addq.b  #1,127(a6)       ; Forbid
1420        move.l  a6,-(a7)         ; a6 retten
1422        movea.l d0,a6            ; a6 -> LibBase
1424        jsr     -12(a6)          ; ---> Expunge
1428        movea.l (a7)+,a6         ; a6 wiederherstellen
142A        jsr     -8A(a6)          ; ---> Permit
142E        rts
1430
1430 ;---------------------------------  OldOpenLibrary
1430
1430 FC1430 moveq   #0,d0            ; d0 := 0
1432        jsr     -228(a6)         ; ---> OpenLibrary
1436        rts
1438
1438 ;---------------------------------  OpenLibrary
1438
1438 FC1438 move.l  d2,-(a7)         ; d2 retten
143A        move.l  d0,d2            ; d2 := Version
143C        addq.b  #1,127(a6)       ; Forbid
1440        lea     17A(a6),a0       ; a0 -> Library List Header
1444 FC1444 bsr     FC165A           ; ---> FindName
1448        tst.l   d0               ; gefunden?
144A        beq.s   FC145E           ; nein: ->
144C        movea.l d0,a0            ; a0 -> LibBase
```

```
144E        cmp.w   14(a0),d2        ; gesuchte > gefundene Version?
1452        bgt.s   FC1444           ; ja: weitersuchen ->
1454        move.l  a6,-(a7)         ; a6 retten
1456        movea.l a0,a6            ; a6 -> LibBase
1458        jsr     -6(a6)           ; ---> Open
145C        movea.l (a7)+,a6         ; a6 wiederherstellen
145E FC145E jsr     -8A(a6)          ; ---> Permit
1462        move.l  (a7)+,d2         ; d2 wiederherstellen
1464        rts
1466
1466 ;--------------------------------- CloseLibrary
1466
1466 FC1466 addq.b  #1,127(a6)       ; Forbid
146A        move.l  a6,-(a7)         ; a6 retten
146C        movea.l a1,a6            ; a6 -> LibBase
146E        jsr     -C(a6)           ; ---> Close
1472        movea.l (a7)+,a6         ; a6 wiederherstellen
1474        jsr     -8A(a6)          ; ---> Permit
1478        rts
147A
147A ;--------------------------------- SetFunction
147A
147A FC147A bset.b  #1,E(a1)         ; lib_Flags: änderungsflag setzen
1480        lea     0(a1,a0.w),a0    ; a0 -> JMP Vektor
1484        move.l  2(a0),-(a7)      ; Vektor auf Stack retten
1488        move.w  #4EF9,(a0)       ; JMP-Kode eintragen
148C        move.l  d0,2(a0)         ; Neuen Vektor dahintersetzen
1490        bsr     FC1498           ; ---> SumLibrary
1494        move.l  (a7)+,d0         ; d0 := alter Vektor
1496        rts
1498
1498 ;--------------------------------- SumLibrary
1498
1498 FC1498 btst.b  #2,E(a1)         ; lib_Flags: Summierflag gesetzt?
149E        beq.s   FC14EA           ; nein: fertig ->
14A0        addq.b  #1,127(a6)       ; Forbid
14A4        bclr.b  #1,E(a1)         ; Änderungsflag rücksetzen
14AA        beq.s   FC14B0           ; war schon rückgesetzt: ->
14AC        clr.w   1C(a1)           ; lib_Sum löschen
14B0 FC14B0 movea.l a1,a0            ; a0 := a1 = LibBase
```

```
14B2        move.w  10(a1),d0          ; d0 := LibNegSize (Bytes)
14B6        lsr.w   #1,d0              ; d0 := LibNegSize (Worte)
14B8        moveq   #0,d1              ; d1 := 0 (Summenregister)
14BA        bra.s   FC14BE             ; --->
14BC
14BC FC14BC add.w   -(a0),d1           ; Prüfsumme der JMP-Liste berechnen
14BE FC14BE dbra    d0,FC14BC
14C2        move.w  1C(a1),d0          ; d0 := lib_Sum
14C6        beq.s   FC14E2             ; = 0: ->
14C8        cmp.w   d0,d1              ; mit berechneter Summe vergleichen
14CA        beq.s   FC14E6             ; gleich: ->
14CC        movem.l d7/a5-a6,-(a7)     ; Register retten
14D0        move.l  #81000003,d7       ; Alert-Kode 'AN_LibChkSum'
14D6        movea.l 4,a6               ; a6 := SysBase
14DA        jsr     -6C(a6)            ; ---> Alert
14DE        movem.l (a7)+,d7/a5-a6     ; Register wiederherstellen
14E2 FC14E2 move.w  d1,1C(a1)          ; Berechnete Summe eintragen
14E6 FC14E6 jsr     -8A(a6)            ; ---> Permit
14EA FC14EA rts
14EC
14EC ;---------------------------------- MakeLibrary
14EC
14EC FC14EC movem.l d2-d7/a2-a3,-(a7)  ; Register retten
14F0        move.l  d0,d2              ; d2 := d0 = Länge des Datenbereichs
14F2        move.l  a0,d4              ; d4 := a0 -> Funktionsvektoren
14F4        move.l  a1,d5              ; d5 := a1 -> InitStruct-Daten
14F6        move.l  a2,d6              ; d6 := a2 -> Init
14F8        move.l  d1,d7              ; d7 := d1 -> SegList
14FA        move.l  a0,d3              ; d3 := a0 -> Funktionsvektoren
14FC        beq.s   FC151E             ; keine Vektoren vorhanden: ->
14FE        movea.l a0,a3              ; a3 := a0 -> Funktionsvektoren
1500        moveq   #FF,d3             ; d3 := -1 = Flag für Relativ-Vektoren
1502        move.l  d3,d0              ; d0 := d3 = -1
1504        cmp.w   (a3),d0            ; Enthält die Tabelle relative Offsets?
1506        bne.s   FC1512             ; nein: ->
1508        addq.l  #2,a3              ; Tabellenzeiger vorrücken
150A FC150A cmp.w   (a3)+,d0           ; Tabellenwort = Endemarke?
150C        dbeq    d3,FC150A          ; nein: d3 dekrementieren ->
1510        bra.s   FC1518             ; --->
1512
```

```
1512 FC1512  cmp.l    (a3)+,d0        ; Tabellen-Langwort = Endemarke?
1514         dbeq     d3,FC1512       ; nein: d3 dekrementieren ->
1518 FC1518  not.w    d3              ; d3 := Zahl der Vektoren
151A         mulu     #6,d3           ; mal 6 =: Länge der Sprungliste
151E FC151E  move.l   d2,d0           ; d0 := DataSize
1520         add.l    d3,d0           ; + Länge der Sprungliste = Gesamtlänge
1522         move.l   #10001,d1       ; Speichertyp Public, Clear
1528         jsr      -C6(a6)         ; ---> AllocMem
152C         tst.l    d0              ; Speicher zugeordnet?
152E         bne.s    FC1536          ; ja: ->
1530         moveq    #0,d0           ; d0 := 0 als Fehlerflag
1532         bra      FC1570          ; ---> Abschluß
1536
1536 FC1536  movea.l  d0,a3           ; a3 -> Anfang des Speicherbereichs
1538         adda.l   d3,a3           ; a3 -> Anfang des Datenbereichs
153A         move.w   d3,10(a3)       ; LibNegSize eintragen
153E         move.w   d2,12(a3)       ; LibPosSize eintragen
1542         movea.l  a3,a0           ; a0 := LibBase
1544         suba.l   a2,a2           ; a2 := 0 = Basis für Adreßberechnung
1546         movea.l  d4,a1           ; a1 := d4 -> Funktionsvektoren
1548         cmpi.w   #FFFF,(a1)      ; Erstes Wort = -1?
154C         bne.s    FC1552          ; nein: absolute Adressen ->
154E         addq.l   #2,a1           ; Tabellenzeiger vorrücken
1550         movea.l  d4,a2           ; a2 := d4 = Basis für Adreßberechnung
1552 FC1552  bsr      FC1576          ; ---> MakeFunctions
1556         tst.l    d5              ; InitStruct-Daten vorhanden?
1558         beq.s    FC1564          ; nein: ->
155A         movea.l  a3,a2           ; a2 := a3 = LibBase
155C         movea.l  d5,a1           ; a1 := d5 -> InitCode
155E         moveq    #0,d0           ; d0 := 0: keine Size-Angabe
1560         jsr      -4E(a6)         ; ---> InitStruct
1564 FC1564  move.l   a3,d0           ; d0 := a3 = LibBase
1566         tst.l    d6              ; InitCode vorhanden?
1568         beq.s    FC1570          ; nein: ->
156A         movea.l  d6,a1           ; a1 := d6 -> InitCode
156C         movea.l  d7,a0           ; a0 := d7 -> SegList
156E         jsr      (a1)            ; ---> Initialisieren
1570 FC1570  movem.l  (a7)+,d2-d7/a2-a3 ; Register wiederherstellen
1574         rts
1576
```

```
1576 ;---------------------------------  MakeFunctions
1576
1576 FC1576  move.l  a3,-(a7)          ; a3 retten
1578         moveq   #0,d0             ; d0 := 0 = Byte-Zähler Anfangswert
157A         move.l  a2,d1             ; d1 := a2 = Offset-/Adressen-Flag
157C         beq.s   FC1594            ; Tabelle enthält Adressen: ->
157E FC157E  move.w  (a1)+,d1          ; nächstes Wort aus Tabelle
1580         cmpi.w  #-1,d1            ; Endemarke?
1584         beq.s   FC15A8            ; ja: ->
1586         lea     0(a2,d1.w),a3     ; a3 := Vektor-Adresse aus Offset
158A         move.l  a3,-(a0)          ; Adresse in Sprungliste eintragen
158C         move.w  #4EF9,-(a0)       ; JMP-Kode davorsetzen
1590         addq.l  #6,d0             ; Bytezähler um Sprunglänge erhöhen
1592         bra.s   FC157E            ; ---> Loop
1594
1594 FC1594  move.l  (a1)+,d1          ; d1 := Langwort aus Tabelle
1596         cmpi.l  #-1,d1            ; Endemarke?
159C         beq.s   FC15A8            ; ja: ->
159E         move.l  d1,-(a0)          ; Adresse in Sprungliste eintragen
15A0         move.w  #4EF9,-(a0)       ; JMP-Kode davorsetzen
15A4         addq.l  #6,d0             ; Bytezähler um Sprunglänge erhöhen
15A6         bra.s   FC1594            ; ---> Loop
15A8
15A8 FC15A8  movea.l (a7)+,a3          ; a3 wiederherstellen
15AA         rts
15AC
15AC ;---------------------------------  Insert
15AC
15AC FC15AC  move.l  a2,d0             ; d0 := a2 -> ListNode (vor Insert)
15AE         beq     FC15D8            ; = 0: AddHead ->
15B2         move.l  (a2),d0           ; d0 := ln_Succ (ListNode)
15B4         beq     FC15C6            ; ListNode = letzter Node in Liste: ->
15B8         movea.l d0,a0             ; a0 := d0 -> ln_Succ (ListNode)
15BA         movem.l d0/a2,(a1)        ; Neuen Node initialisieren
15BE         move.l  a1,4(a0)          ; Neuen ln_Pred eintragen
15C2         move.l  a1,(a2)           ; Neuen ln_Succ eintragen
15C4         rts
15C6
15C6 FC15C6  move.l  a2,(a1)           ; Ist ListNode der letzte Node in
15C8         movea.l 4(a2),a0          ; der Liste, so erfolgt der Insert
```

```
15CC        move.l  a0,4(a1)         ; vor ListNode
15D0        move.l  a1,4(a2)
15D4        move.l  a1,(a0)
15D6        rts
15D8
15D8 ;--------------------------------  AddHead
15D8
15D8 FC15D8 move.l  (a0),d0          ; Neuer Node wird eingefügt zwischen
15DA        move.l  a1,(a0)          ; ListHeader und erstem Node
15DC        movem.l d0/a0,(a1)
15E0        movea.l d0,a0
15E2        move.l  a1,4(a0)
15E6        rts
15E8
15E8 ;--------------------------------  AddTail
15E8
15E8 FC15E8 lea     4(a0),a0         ; a0 -> lh_Tail
15EC        move.l  4(a0),d0         ; d0 := alter lh_TailPred
15F0        move.l  a1,4(a0)         ; Neuen lh_TailPred eintragen
15F4        move.l  a0,(a1)          ; Neuen ln_Succ eintragen
15F6        move.l  d0,4(a1)         ; Neuen ln_Pred eintragen
15FA        movea.l d0,a0            ; a0 := alter lh_TailPred
15FC        move.l  a1,(a0)          ; TailPred -> neuer Node
15FE        rts
1600
1600 ;--------------------------------  Remove
1600
1600 FC1600 movea.l (a1),a0          ; a0 := ln_Succ
1602        movea.l 4(a1),a1         ; a1 := ln_Pred
1606        move.l  a0,(a1)          ; ln_Succ in Vorgänger eintragen
1608        move.l  a1,4(a0)         ; ln_Pred in Nachfolger eintragen
160C        rts
160E
160E ;--------------------------------  RemHead
160E
160E FC160E movea.l (a0),a1          ; a1 -> 1. Node in der Liste
1610        move.l  (a1),d0          ; d0 -> 2. Node in der Liste
1612        beq.s   FC161C           ; Liste leer: ->
1614        move.l  d0,(a0)          ; lh_Head -> 2. Node
1616        exg     d0,a1            ; a1 -> 2. Node
```

```
1618        move.l  a0,4(a1)          ; ln_Pred -> ListHeader
161C FC161C rts
161E
161E ;---------------------------------  RemTail
161E
161E FC161E movea.l 8(a0),a1          ; a1 := lh_TailPred -> letzter Node
1622        move.l  4(a1),d0          ; d0 := ln_Pred -> vorletzter Node
1626        beq.s   FC1632            ; Liste leer: ->
1628        move.l  d0,8(a0)          ; lh_TailPred -> vorletzter Node
162C        exg     d0,a1             ; a1 -> vorletzter Node
162E        move.l  a0,(a1)           ; ln_Succ eintragen
1630        addq.l  #4,(a1)
1632 FC1632 rts
1634
1634 ;---------------------------------  Enqueue
1634
1634 FC1634 move.b  9(a1),d1          ; d1 := Priorität
1638        move.l  (a0),d0           ; d0 := lh_Head
163A FC163A movea.l d0,a0             ; a0 := d0
163C        move.l  (a0),d0           ; d0 := ln_Succ
163E        beq.s   FC1646            ; Ende der Liste: ->
1640        cmp.b   9(a0),d1          ; Priorität kleiner?
1644        ble.s   FC163A            ; ja: weitersuchen ->
1646 FC1646 move.l  4(a0),d0          ; AddNode (vgl. AddTail)
164A        move.l  a1,4(a0)
164E        move.l  a0,(a1)
1650        move.l  d0,4(a1)
1654        movea.l d0,a0
1656        move.l  a1,(a0)
1658        rts
165A
165A ;---------------------------------  FindName
165A
165A FC165A move.l  a2,-(a7)          ; a2 retten
165C        movea.l a0,a2             ; a2 := a0 -> Suchanfang (Node/Header)
165E        move.l  a1,d1             ; d1 := a1 -> Namensstring
1660        move.l  (a2),d0           ; d0 := ln_Succ bzw. lh_Head
1662        beq.s   FC167C            ; Liste leer: d0 = 0 ->
1664 FC1664 movea.l d0,a2             ; a2 := ln_Succ
1666        move.l  (a2),d0           ; d0 := nächster ln_Succ
```

```
1668         beq.s    FC167C          ; Ende der Liste: dØ = Ø ->
166A         movea.l A(a2),aØ         ; aØ := ln_Name aus Node
166E         movea.l d1,a1            ; a1 -> gesuchter Namensstring
167Ø FC167Ø  cmpm.b  (aØ)+,(a1)+      ; zeichenweise vergleichen
1672         bne.s    FC1664          ; ungleich: weitersuchen ->
1674         tst.b    -1(aØ)          ; Endemarke erreicht?
1678         bne.s    FC167Ø          ; nein: Vergleich fortsetzen ->
167A         move.l  a2,dØ            ; dØ := a2 -> Node gleichen Namens
167C FC167C  movea.l d1,a1            ; a1 := d1 -> Namensstring
167E         movea.l (a7)+,a2         ; a2 wiederherstellen
168Ø         rts
1682
1682 ;------ Forbid, Enqueue, Permit
1682
1682 FC1682  addq.b  #1,127(a6)       ; Forbid
1686         bsr.s    FC1634          ; ---> Enqueue
1688         jsr      -8A(a6)         ; ---> Permit
168C         rts
168E
168E ;------ Forbid, Remove, Permit
168E
168E FC168E  addq.b  #1,127(a6)       ; Forbid
1692         bsr      FC16ØØ          ; ---> Remove
1696         jsr      -8A(a6)         ; ---> Permit
169A         rts
169C
169C ;---------------------------------  Allocate
169C
169C FC169C  tst.l    dØ              ; Angeforderte Bytezahl
169E         beq.s    FC16EC          ; = Ø: fertig ->
16AØ         movem.l d3/a2-a3,-(a7)   ; Register retten
16A4         addq.l  #7,dØ            ; Bytezahl modulo 8 aufrunden
16A6         andi.b  #F8,dØ
16AA         moveq    #Ø,d3           ; für Fehlermeldung
16AC         cmp.l    1C(aØ),dØ       ; Anforderung > freier Speicher?
16BØ         bhi      FC16E6          ; ja: ->
16B4         lea      1Ø(aØ),a2       ; a2 -> mh_First = erster freier Block
16B8 FC16B8  move.l  (a2),d3          ; d3 := Link zum nächsten Block
16BA         beq.s    FC16E6          ; Liste zu Ende: ->
16BC         movea.l d3,a1            ; a1 -> nächster Blockanfang
```

```
16BE        cmp.l    4(a1),dØ          ; Blocklänge > angeforderte Bytezahl?
16C2        bls.s    FC16C8            ; ja: ->
16C4        movea.l  a1,a2             ; a2 := a1 -> nächster Blockanfang
16C6        bra.s    FC16B8            ; ---> Loop
16C8
16C8 FC16C8 beq.s    FC16DE            ; Blocklänge = Anforderung: ->
16CA        lea      Ø(a1,dØ.l),a3     ; a3 -> Anfang des Restblocks
16CE        move.l   (a1),(a3)         ; Link zum nächsten Block eintragen
16DØ        move.l   4(a1),d3          ; d3 := Länge des freien Blocks
16D4        sub.l    dØ,d3             ; - angeforderte Bytezahl
16D6        move.l   d3,4(a3)          ; als neue Blocklänge eintragen
16DA        move.l   a3,(a2)           ; Neuen Blockanfang in Vorblock-Link
16DC        bra.s    FC16EØ            ; --->
16DE
16DE FC16DE move.l   (a1),(a2)         ; Link zurückkopieren
16EØ FC16EØ sub.l    dØ,1C(aØ)         ; mh_Free um Anforderung vermindern
16E4        move.l   a1,d3             ; d3 := a1 -> Allozierter Bereich
16E6 FC16E6 move.l   d3,dØ             ; dØ := d3 -> Allozierter Bereich
16E8        movem.l  (a7)+,d3/a2-a3    ; Register wiederherstellen
16EC FC16EC rts
16EE
16EE ;------ Speicherfehler
16EE
16EE FC16EE movem.l  d7/a5-a6,-(a7)    ; Register retten
16F2        move.l   #81ØØØØØ5,d7      ; Alert-Kode 'AN_MemCorrupt'
16F8        movea.l  4,a6              ; a6 := SysBase
16FC        jsr      -6C(a6)           ; ---> Alert
17ØØ        movem.l  (a7)+,d7/a5-a6    ; Register wiederherstellen
17Ø4
17Ø4 ;---------------------------------- Deallocate
17Ø4
17Ø4 FC17Ø4 tst.l    dØ                ; Zahl der freizugebenden Bytes
17Ø6        beq.s    FC177C            ; = Ø: fertig ->
17Ø8        movem.l  d3/a2,-(a7)       ; Register retten
17ØC        move.l   a1,d1             ; d1 := a1 -> freizugebender Block
17ØE        moveq    #F8,d3            ; Blockadresse modulo 8 abrunden
171Ø        and.l    d3,d1
1712        exg      d1,a1             ; a1 -> Blockanfang abgerundet
1714        sub.l    a1,d1             ; Differenz bei Rundung
1716        add.l    d1,dØ             ; zur Länge addieren
```

```
1718          addq.l  #7,d0         ; Ergebnis modulo 8 aufrunden
171A          and.l   d3,d0
171C          beq     FC1778        ; Ergebnis = 0: ->
1720          lea     10(a0),a2     ; a2 := mh_First
1724          move.l  (a2),d3       ; d3 := Block Link
1726          beq.s   FC1748        ; Ende der Liste: ->
1728 FC1728   cmpa.l  d3,a1         ; Anfang des freizugebenden Blocks
172A          bcs.s   FC1734        ; < Anfang des Speicherblocks: ->
172C          beq.s   FC177E        ; = Anfang des Speicherblocks: Fehler ->
172E          movea.l d3,a2         ; a2 := Link zum nächsten Block
1730          move.l  (a2),d3       ; d3 := neuer Link
1732          bne.s   FC1728        ; Liste nicht zu Ende: weitersuchen ->
1734 FC1734   moveq   #10,d1        ; d1 := 16
1736          add.l   a0,d1         ;
1738          cmp.l   a2,d1
173A          beq.s   FC1748
173C          move.l  4(a2),d3      ; d3 := Länge des freien Blocks
1740          add.l   a2,d3         ; + Blockanfang = Blockende
1742          cmp.l   a1,d3         ; Anfang des freizugebenden Bereichs
1744          beq.s   FC1752        ; = Ende des freien Blocks: ->
1746          bhi.s   FC177E        ; liegt im freien Bereich: Fehler ->
1748 FC1748   move.l  (a2),(a1)     ; Neuen Blockheader erzeugen
174A          move.l  a1,(a2)
174C          move.l  d0,4(a1)
1750          bra.s   FC1758        ; --->
1752
1752 FC1752   add.l   d0,4(a2)      ; Blocklänge um Freibytes vergrößern
1756          movea.l a2,a1         ; a1 := a2 -> Blockanfang
1758 FC1758   tst.l   (a1)          ; Link am Blockanfang eingetragen?
175A          beq.s   FC1774        ; nein: ->
175C          move.l  4(a1),d3      ; d3 := Blocklänge
1760          add.l   a1,d3         ; + Blockanfang = Blockende+1
1762          cmp.l   (a1),d3       ; mit Linkadresse vergleichen
1764          bhi.s   FC177E        ; Linkadresse kleiner: Fehler ->
1766          bne.s   FC1774        ; Linkadresse größer: ->
1768          movea.l (a1),a2       ; Nächsten Block einlinken
176A          move.l  (a2),(a1)
176C          move.l  4(a2),d3
1770          add.l   d3,4(a1)
1774 FC1774   add.l   d0,1C(a0)     ; mh_Free um Freibytes erhöhen
```

```
1778 FC1778  movem.l (a7)+,d3/a2        ; Register wiederherstellen
177C FC177C  rts
177E
177E ;------ Freigabe-Fehler
177E
177E FC177E  movem.l d7/a5-a6,-(a7)     ; Register retten
1782         move.l  #81000009,d7       ; Alert-Kode 'AN_FreeTwice'
1788         movea.l 4,a6               ; a6 := SysBase
178C         jsr     -6C(a6)            ; ---> Alert
1790         movem.l (a7)+,d7/a5-a6     ; Register wiederherstellen
1794
1794 ;---------------------------------  AllocMem
1794
1794 FC1794  addq.b  #1,127(a6)         ; Forbid
1798         movem.l d2-d3/a2,-(a7)     ; Register retten
179C         move.l  d0,d3              ; d3 := d0 = angeforderte Bytezahl
179E         move.l  d1,d2              ; d2 := d1 = Speichertyp
17A0         lea     142(a6),a2         ; a2 -> Memory List Header
17A4 FC17A4  movea.l (a2),a2            ; a2 -> 1. Node
17A6         tst.l   (a2)               ; Liste leer?
17A8         bne.s   FC17AE             ; nein: ->
17AA         moveq   #0,d0              ; d0 := 0 als Fehlerkode
17AC         bra.s   FC17E6             ; --->
17AE
17AE FC17AE  move.w  E(a2),d0           ; d0 := Speicher-Attribut
17B2         and.w   d2,d0              ; entspricht der Anforderung?
17B4         cmp.w   d2,d0
17B6         bne.s   FC17A4             ; nein: weitersuchen ->
17B8         movea.l a2,a0              ; a0 := a2 -> Node
17BA         move.l  d3,d0              ; d0 := d3 = angeforderte Bytezahl
17BC         bsr     FC169C             ; ---> Allocate
17C0         tst.l   d0                 ; Speicher zugeordnet?
17C2         beq.s   FC17A4             ; nein: weitersuchen ->
17C4         btst.l  #10,d2             ; Speichertyp 'Clear'?
17C8         beq.s   FC17E6             ; nein: ->
17CA         moveq   #0,d1              ; Löschkode 0
17CC         addq.l  #3,d3              ; Bytezahl+3
17CE         lsr.l   #2,d3              ; dividiert durch 4 ergibt Langworte
17D0         movea.l d0,a0              ; a0 := d0 -> Bereichsanfang
17D2         bra.s   FC17D6             ; --->
```

```
17D4
17D4 FC17D4  move.l  d1,(a0)+          ; Speicherbereich löschen
17D6 FC17D6  dbra    d3,FC17D4         ; bis Zähler d3.w = 0
17DA         swap    d3                ; H-Wort holen
17DC         tst.w   d3                ; ist es Null?
17DE         beq.s   FC17E6            ; ja: fertig ->
17E0         subq.w  #1,d3             ; H-Wort dekrementieren
17E2         swap    d3                ; und zurücktauschen
17E4         bra.s   FC17D4            ; ---> Loop
17E6
17E6 FC17E6  jsr     -8A(a6)           ; ---> Permit
17EA         movem.l (a7)+,d2-d3/a2    ; Register wiederherstellen
17EE         rts
17F0
17F0 ;---------------------------------  FreeMem
17F0
17F0 FC17F0  addq.b  #1,127(a6)        ; Forbid
17F4         tst.l   d0                ; Freizugebende Bytezahl
17F6         beq.s   FC1814            ; = 0: fertig ->
17F8         lea     142(a6),a0        ; a0 -> Memory List Header
17FC FC17FC  movea.l (a0),a0           ; a0 -> 1. Node
17FE         tst.l   (a0)              ; Liste leer?
1800         beq     FC16EE            ; ja: Fehler ->
1804         cmpa.l  14(a0),a1         ; FreeMem-Anfang
1808         bcs.s   FC17FC            ; < Anfang der Speicherregion: ->
180A         cmpa.l  18(a0),a1
180E         bcc.s   FC17FC            ; > Ende der Speicherregion: ->
1810         bsr     FC1704            ; ---> Deallocate
1814 FC1814  jsr     -8A(a6)           ; ---> Permit
1818         rts
181A
181A ;---------------------------------  TypeOfMem
181A
181A FC181A  addq.b  #1,127(a6)        ; Forbid
181E         lea     142(a6),a0        ; a0 -> Memory List Header
1822         moveq   #0,d0             ; d0 := 0
1824 FC1824  movea.l (a0),a0           ; a0 -> 1. Node
1826         tst.l   (a0)              ; Ende der Liste?
1828         beq.s   FC183A            ; ja: ->
182A         cmpa.l  14(a0),a1         ; Block-Anfang
```

```
182E        bcs.s    FC1824           ; < Anfang der Speicherregion: ->
1830        cmpa.l   18(a0),a1
1834        bcc.s    FC1824           ; > Ende der Speicherregion: ->
1836        move.w   E(a0),d0         ; d0 := mh_Attributes = Speichertyp
183A FC183A jsr      -8A(a6)          ; ---> Permit
183E        rts
1840
1840 ;---------------------------------  AllocAbs
1840
1840 FC1840 addq.b   #1,127(a6)       ; Forbid
1844        movem.l d2-d4/a2-a3,-(a7) ; Register retten
1848        move.l   a1,d2            ; d2 := a1 = Anfang des Blocks
184A        andi.l   #7,d2            ; modulo 8
1850        suba.l   d2,a1            ; Blockanfang modulo 8 abrunden
1852        add.l    d2,d0            ; d0 := Bytezahl + Blockanfang
1854        addq.l   #7,d0            ; modulo 8 aufrunden
1856        andi.b   #F8,d0           ; ergibt Blockende+1
185A        lea      142(a6),a0       ; a0 -> Memory List Header
185E FC185E movea.l (a0),a0           ; a0 -> Memory Region Header
1860        tst.l    (a0)             ; Region leer?
1862        beq.s    FC18CC           ; ja: ->
1864        cmpa.l   14(a0),a1        ; Blockanfang
1868        bcs.s    FC185E           ; < Anfang der Speicherregion: ->
186A        cmpa.l   18(a0),a1
186E        bcc.s    FC185E           ; > Ende der Speicherregion: ->
1870        cmp.l    1C(a0),d0        ; Bytezahl > mh_Free?
1874        bhi.s    FC18CC           ; ja: ->
1876        movea.l a1,a3             ; a3 := a1 = Anfang des Bereichs
1878        move.l   a1,d2            ; d2 := a1
187A        add.l    d0,d2            ; + Bytezahl = Ende des Bereichs
187C        lea      10(a0),a2        ; a2 := mh_First
1880 FC1880 move.l   (a2),d3          ; d3 -> nächster freier Block
1882        beq.s    FC18CC           ; Liste zu Ende: ->
1884        movea.l d3,a1             ; a1 := d3 -> Blockanfang
1886        move.l   4(a1),d4         ; d4 := Länge des Blocks
188A        add.l    d3,d4            ; + Blockanfang = Blockende+1
188C        cmp.l    d2,d4            ; Blockende < Ende der Anforderung?
188E        bcc.s    FC1894           ; nein: ->
1890        movea.l a1,a2             ; a2 := a1 -> Blockanfang
1892        bra.s    FC1880           ; ---> Loop
```

```
1894
1894 FC1894  cmp.l    a3,d3                ; Blockanfang > Anfang der Anforderung?
1896         bhi.s    FC18CC               ; ja: keine Allokation möglich ->
1898         sub.l    d0,1C(a0)            ; mh_Free um Byte-Anforderung vermindern
189C         sub.l    d2,d4                ; d4 := Blockende - Bereichsende
189E         bne.s    FC18A4               ; nicht gleich: ->
18A0         movea.l  (a1),a0              ; a0 := Linkadresse
18A2         bra.s    FC18B0               ; --->
18A4
18A4 FC18A4  lea      0(a3,d0.l),a0        ; a0 -> Bereichsende+1
18A8         move.l   (a1),(a0)            ; Link kopieren
18AA         move.l   a0,(a1)              ; Adresse in Link des Vorblocks
18AC         move.l   d4,4(a0)             ; Länge des Restblocks eintragen
18B0 FC18B0  cmp.l    a3,d3                ; Blockanfang = Anfang der Anforderung?
18B2         beq.s    FC18BE               ; ja: ->
18B4         sub.l    a3,d3                ; Länge des Zwischenblocks berechnen
18B6         neg.l    d3
18B8         move.l   d3,4(a1)             ; und in Blockheader eintragen
18BC         bra.s    FC18C0               ; --->
18BE
18BE FC18BE  move.l   a0,(a2)              ; Link eintragen
18C0 FC18C0  move.l   a3,d0                ; d0 := Blockanfang
18C2 FC18C2  movem.l  (a7)+,d2-d4/a2-a3    ; Register wiederherstellen
18C6         jsr      -8A(a6)              ; ---> Permit
18CA         rts
18CC
18CC FC18CC  moveq    #0,d0                ; d0 := 0
18CE         bra.s    FC18C2               ; --->
18D0
18D0 ;---------------------------------  AvailMem
18D0
18D0 FC18D0  movem.l  d3/a2,-(a7)          ; Register retten
18D4         moveq    #0,d3                ; d3 := 0
18D6         lea      142(a6),a1           ; a1 -> Memory List Header
18DA         addq.b   #1,127(a6)           ; Forbid
18DE FC18DE  movea.l  (a1),a1              ; a1 -> Memory Region Header
18E0         tst.l    (a1)                 ; Ende der Liste?
18E2         beq.s    FC1912               ; ja: ->
18E4         move.w   E(a1),d0             ; d0 := mh_Attributes
18E8         and.w    d1,d0                ; mit d1 maskieren
```

```
18EA        cmp.w    d1,dØ              ; richtiger Speichertyp?
18EC        bne.s    FC18DE             ; nein: weitersuchen ->
18EE        btst.l   #11,d1             ; Speichertyp 'largest' verlangt?
18F2        bne.s    FC18FA             ; ja: ->
18F4        add.l    1C(a1),d3          ; mh_Free zu d3 addieren
18F8        bra.s    FC18DE             ; ---> Loop
18FA
18FA FC18FA move.l   1Ø(a1),dØ          ; dØ -> mh_First
18FE FC18FE beq.s    FC18DE             ; Liste zu Ende: ->
19ØØ        movea.l  dØ,aØ              ; aØ -> Speicherblock
19Ø2        cmp.l    4(aØ),d3           ; Bytes in Region > d3?
19Ø6        bge.s    FC19ØE             ; nein: ->
19Ø8        movea.l  aØ,a2              ; a2 := aØ -> Speicherblock
19ØA        move.l   4(aØ),d3           ; d3 := Bytezahl im Speicherblock
19ØE FC19ØE move.l   (aØ),dØ            ; dØ := Link zum nächsten Block
191Ø        bra.s    FC18FE             ; ---> Loop
1912
1912 FC1912 jsr      -8A(a6)            ; ---> Permit
1916        move.l   d3,dØ              ; dØ := d3 = Ergebnis
1918        movem.l  (a7)+,d3/a2        ; Register wiederherstellen
191C        rts
191E
191E ;--------------------------------- AllocEntry
191E
191E FC191E movem.l  d2-d3/a2-a3,-(a7)  ; Register retten
1922        movea.l  aØ,a2              ; a2 := aØ = Memory Liste
1924        moveq    #Ø,d2              ; d2 := Ø = Zähler der Einträge
1926        moveq    #Ø,d3              ; d3 := Ø = Gesamtzahl der Einträge
1928        move.w   E(a2),d3           ; d3 := Anzahl der Listeneinträge
192C        move.l   d3,dØ              ; dØ := d3
192E        lsl.l    #3,dØ              ; mal 8 = Länge der Liste
193Ø        addi.l   #1Ø,dØ             ; + 14 (für Node) + 2 (für NumEntries)
1936        move.l   #1ØØØØ,d1          ; d1 := Speichertyp 'Clear'
193C        jsr      -C6(a6)            ; ---> AllocMem für Memory Liste
194Ø        tst.l    dØ                 ; Speicher zugeordnet?
1942        beq.s    FC19A2             ; nein: ->
1944        movea.l  dØ,a3              ; a3 := Anfang des Speicherbereichs
1946        move.w   d3,E(a3)           ; ml_NumEntries eintragen
194A        lea      1Ø(a2),a2          ; a2 -> 1. Eintrag in eingeg. Liste
194E        lea      1Ø(a3),a3          ; a3 -> 1. Eintrag in angelegt. Liste
```

```
1952 FC1952 move.l  0(a2),d1             ; d1 := angeforderter Speichertyp
1956        move.l  4(a2),d0             ; d0 := angeforderte Bytezahl
195A        move.l  d0,4(a3)             ; in angelegte Liste eintragen
195E        beq.s   FC1968               ; Bytezahl = 0: fertig ->
1960        jsr     -C6(a6)              ; ---> AllocMem
1964        tst.l   d0                   ; Speicher zugeordnet?
1966        beq.s   FC198A               ; nein: ->
1968 FC1968 move.l  d0,0(a3)             ; Adresse des Speicherblocks eintragen
196C        addq.l  #8,a2                ; a2 und
196E        addq.l  #8,a3                ; a3 auf nächsten Eintrag vorrücken
1970        addq.l  #1,d2                ; Zähler inkrementieren
1972        cmp.l   d2,d3                ; Alle Einträge bearbeitet?
1974        bgt.s   FC1952               ; nein: weitermachen ->
1976        lsl.l   #3,d2                ; Anzahl der Einträge * 8 = Länge
1978        neg.l   d2                   ; negieren
197A        lea     -10(a2,d2.l),a2      ; a2 -> Anfang der eingegebenen Liste
197E        lea     -10(a3,d2.l),a3      ; a3 -> Anfang der angelegten Liste
1982        move.l  a3,d0                ; d0 := a3
1984 FC1984 movem.l (a7)+,d2-d3/a2-a3    ; Register wiederherstellen
1988        rts
198A
198A ;------ Kein Platz für Anforderung
198A
198A FC198A move.l  d1,d3                ; d3 := d1 = angeforderter Speichertyp
198C FC198C tst.l   d2                   ; bearbeitete Einträge vorhanden?
198E        beq.s   FC19A4               ; nein: ->
1990        subq.l  #1,d2                ; Zähler dekrementieren
1992        subq.l  #8,a3                ; Zeiger auf vorigen Eintrag zurück
1994        movea.l 0(a3),a1             ; a1 -> zugeordneter Speicher
1998        move.l  4(a3),d0             ; d0 := Anzahl der Bytes
199C        jsr     -D2(a6)              ; ---> FreeMem gibt Speicher wieder frei
19A0        bra.s   FC198C               ; ---> Loop
19A2
19A2 ;------ Kein Platz für Memory Liste
19A2
19A2 FC19A2 move.l  d1,d3                ; d3 := d1
19A4 FC19A4 move.l  d3,d0                ; d0 := d3 = Speichertyp bei Fehler
19A6        bset.l  #1F,d0               ; Bit 31 = Fehlerflag
19AA        bra.s   FC1984               ; --->
19AC
```

```
19AC ;---------------------------------  FreeEntry
19AC
19AC FC19AC  movem.l d2/a2-a3,-(a7)     ; Register retten
19B0         movea.l a0,a2              ; a2 := a0 -> Memory Liste
19B2         lea     10(a2),a3          ; a3 -> 1. Eintrag in der Liste
19B6         move.w  E(a2),d2           ; d2 := Anzahl der Einträge
19BA         bra.s   FC19CC             ; --->
19BC
19BC FC19BC  movea.l 0(a3),a1           ; a1 -> Anfang des Speicherblocks
19C0         move.l  4(a3),d0           ; d0 := Anzahl der Bytes
19C4         beq.s   FC19CA             ; = 0: ->
19C6         jsr     -D2(a6)            ; ---> FreeMem für Eintrag
19CA FC19CA  addq.l  #8,a3              ; a3 auf nächsten Eintrag vorrücken
19CC FC19CC  dbra    d2,FC19BC          ; bis letzter Eintrag bearbeitet: ->
19D0         moveq   #0,d0              ; d0.1 := 0
19D2         move.w  E(a2),d0           ; d0 := Anzahl der Einträge
19D6         lsl.l   #3,d0              ; mal 8 = Länge aller Einträge
19D8         addi.l  #10,d0             ; + 14 (Node) + (2) NumEntries
19DE         movea.l a2,a1              ; a1 := a2 -> Memory Liste
19E0         jsr     -D2(a6)            ; ---> FreeMem für Liste
19E4         movem.l (a7)+,d2/a2-a3     ; Register wiederherstellen
19E8         rts
19EA
19EA ;---------------------------------  AddMemList
19EA
19EA FC19EA  move.l  a1,A(a0)           ; ln_Name
19EE         lea     20(a0),a1          ; a1 zeigt hinter den Mem.Header
19F2         move.b  #A,8(a0)           ; ln_Type
19F8         move.b  d2,9(a0)           ; ln_Pri
19FC         move.w  d1,E(a0)           ; mh_Attributes
1A00         move.l  a1,d1              ; d1 := a1
1A02         addq.l  #7,d1              ; d1 auf nächste
1A04         andi.b  #F8,d1             ; Langwortgrenze ausrichten
1A08         exg     d1,a1              ; d1 mit a1 vertauschen
1A0A         sub.l   a1,d1              ; Differenz: -8 < d1 <= 0
1A0C         add.l   d1,d0              ; Freispeicher entspr.verkleinern
1A0E         andi.b  #F8,d0             ; und auf Langwortgrenze abrunden
1A12         subi.l  #20,d0             ; Länge des Memory Headers abziehen
1A18         move.l  a1,10(a0)          ; mh_First: Erstes freies Stück
1A1C         move.l  a1,14(a0)          ; mh_Lower: Anfang des Freispeichers
```

```
1A20        move.l  a1,d1           ; d1 := mh_Lower
1A22        add.l   d0,d1           ; Länge des Freispeichers addieren
1A24        move.l  d1,18(a0)       ; ergibt mh_Upper
1A28        move.l  d0,1C(a0)       ; mh_Free
1A2C        move.l  d0,4(a1)        ; mc_Bytes in Speicherstück eintragen
1A30        clr.l   (a1)            ; mc_Next := 0: letztes Stück
1A32        movea.l a0,a1           ; a1 := Adr. des Memory Region Headers
1A34        lea     142(a6),a0      ; a0 := Adr. des MemList Headers
1A38        bsr     FC1682          ; ---> Forbid, Enqueue, Permit
1A3C        rts
1A3E
1A3E        DC.W    0
1A40
1A40 ;---------------------------------  Exec Library Offset Tabelle
1A40
1A40 FC1A40 DC.W    08A0            ; FC22E0  Open
1A42        DC.W    08A8            ; FC22E8  Close
1A44        DC.W    08AC            ; FC22EC  Expunge
1A46        DC.W    08AC            ; FC22EC  Extfunct
1A48        DC.W    EE6A            ; FC08AA  Supervisor
1A4A        DC.W    F420            ; FC0E60  ExitIntr
1A4C        DC.W    F446            ; FC0E86  Schedule
1A4E        DC.W    04F8            ; FC1F38  Reschedule
1A50        DC.W    F4A0            ; FC0EE0  Switch
1A52        DC.W    F4EA            ; FC0F2A  Dispatch
1A54        DC.W    F58E            ; FC0FCE  Exception
1A56        DC.W    F0B0            ; FC0AF0  InitCode
1A58        DC.W    F188            ; FC0BC8  InitStruct
1A5A        DC.W    FAAC            ; FC14EC  MakeLibrary
1A5C        DC.W    FB36            ; FC1576  MakeFunctions
1A5E        DC.W    F080            ; FC0AC0  FindResident
1A60        DC.W    F0E8            ; FC0B28  InitResident
1A62        DC.W    1596            ; FC2FD6  Alert
1A64        DC.W    08EE            ; FC232E  Debug
1A66        DC.W    F9AC            ; FC13EC  Disable
1A68        DC.W    F9BA            ; FC13FA  Enable
1A6A        DC.W    051A            ; FC1F5A  Forbid
1A6C        DC.W    0520            ; FC1F60  Permit
1A6E        DC.W    F6E2            ; FC1122  SetSR
1A70        DC.W    F708            ; FC1148  SuperState
```

```
1A72        DC.W    F734            ; FC1174  UserState
1A74        DC.W    F74E            ; FC118E  SetIntVector
1A76        DC.W    F794            ; FC11D4  AddIntServer
1A78        DC.W    F7D4            ; FC1214  RemIntServer
1A7A        DC.W    F8E0            ; FC1320  Cause
1A7C        DC.W    FC5C            ; FC169C  Allocate
1A7E        DC.W    FCC4            ; FC1704  Deallocate
1A80        DC.W    FD54            ; FC1794  AllocMem
1A82        DC.W    FE00            ; FC1840  AllocAbs
1A84        DC.W    FDB0            ; FC17F0  FreeMem
1A86        DC.W    FE90            ; FC18D0  AvailMem
1A88        DC.W    FEDE            ; FC191E  AllocEntry
1A8A        DC.W    FF6C            ; FC19AC  FreeEntry
1A8C        DC.W    FB6C            ; FC15AC  Insert
1A8E        DC.W    FB98            ; FC15D8  AddHead
1A90        DC.W    FBA8            ; FC15E8  AddTail
1A92        DC.W    FBC0            ; FC1600  Remove
1A94        DC.W    FBCE            ; FC160E  RemHead
1A96        DC.W    FBDE            ; FC161E  RemTail
1A98        DC.W    FBF4            ; FC1634  Enqueue
1A9A        DC.W    FC1A            ; FC165A  FindName
1A9C        DC.W    0208            ; FC1C48  AddTask
1A9E        DC.W    02B4            ; FC1CF4  RemTask
1AA0        DC.W    0334            ; FC1D74  FindTask
1AA2        DC.W    0388            ; FC1DC8  SetTaskPri
1AA4        DC.W    03E2            ; FC1E22  SetSignal
1AA6        DC.W    03D8            ; FC1E18  SetExcept
1AA8        DC.W    0490            ; FC1ED0  Wait
1AAA        DC.W    0408            ; FC1E48  Signal
1AAC        DC.W    0584            ; FC1FC4  AllocSignal
1AAE        DC.W    05BC            ; FC1FFC  FreeSignal
1AB0        DC.W    054E            ; FC1F8E  AllocTrap
1AB2        DC.W    0574            ; FC1FB4  FreeTrap
1AB4        DC.W    00D8            ; FC1B18  AddPort
1AB6        DC.W    00F0            ; FC1B30  RemPort
1AB8        DC.W    00F4            ; FC1B34  PutMsg
1ABA        DC.W    016E            ; FC1BAE  GetMsg
1ABC        DC.W    019C            ; FC1BDC  ReplyMsg
1ABE        DC.W    01B6            ; FC1BF6  WaitPort
1AC0        DC.W    01DE            ; FC1C1E  FindPort
```

```
1AC2    DC.W   F9CC      ; FC140C AddLibrary
1AC4    DC.W   F9DA      ; FC141A RemLibrary
1AC6    DC.W   F9F0      ; FC1430 OldOpenLibrary
1AC8    DC.W   FA26      ; FC1466 CloseLibrary
1ACA    DC.W   FA3A      ; FC147A SetFunction
1ACC    DC.W   FA58      ; FC1498 SumLibrary
1ACE    DC.W   EC14      ; FC0654 AddDevice
1AD0    DC.W   EC22      ; FC0662 RemDevice
1AD2    DC.W   EC26      ; FC0666 OpenDevice
1AD4    DC.W   EC74      ; FC06B4 CloseDevice
1AD6    DC.W   EC9C      ; FC06DC DoIO
1AD8    DC.W   EC8A      ; FC06CA SendIO
1ADA    DC.W   ED0E      ; FC074E CheckIO
1ADC    DC.W   ECB2      ; FC06F2 WaitIO
1ADE    DC.W   ED2A      ; FC076A AbortIO
1AE0    DC.W   01E8      ; FC1C28 AddResource
1AE2    DC.W   01F0      ; FC1C30 RemResource
1AE4    DC.W   01F4      ; FC1C34 OpenResource
1AE6    DC.W   07B8      ; FC21F8 RawIOInit
1AE8    DC.W   07C2      ; FC2202 RawMayGetChar
1AEA    DC.W   07EE      ; FC222E RawPutChar
1AEC    DC.W   06A8      ; FC20E8 RawDoFmt
1AEE    DC.W   F700      ; FC1140 GetCC
1AF0    DC.W   FDDA      ; FC181A TypeOfMem
1AF2    DC.W   131C      ; FC2D5C Procure
1AF4    DC.W   1332      ; FC2D72 Vacate
1AF6    DC.W   F9F8      ; FC1438 OpenLibrary
1AF8    DC.W   1354      ; FC2D94 InitSemaphore
1AFA    DC.W   1374      ; FC2DB4 ObtainSemaphore
1AFC    DC.W   13C4      ; FC2E04 ReleaseSemaphore
1AFE    DC.W   1428      ; FC2E68 AttemptSemaphore
1B00    DC.W   1458      ; FC2E98 ObtainSemaphoreList
1B02    DC.W   14CE      ; FC2F0E ReleaseSemaphoreList
1B04    DC.W   14F4      ; FC2F34 FindSemaphore
1B06    DC.W   14E4      ; FC2F24 AddSemaphore
1B08    DC.W   14F0      ; FC2F30 RemSemaphore
1B0A    DC.W   EFFC      ; FC0A3C SumKickData
1B0C    DC.W   FFAA      ; FC19EA AddMemList
1B0E    DC.W   1504      ; FC2F44 CopyMem
1B10    DC.W   1500      ; FC2F40 CopyMemQuick
```

```
1B12         DC.W    FFFF               ; Endemarke
1B14         DC.W    FFFF
1B16         DC.W    Ø
1B18
1B18 ;---------------------------------  AddPort
1B18
1B18 FC1B18  lea     14(a1),aØ          ; aØ -> Message List Header im Msg Port
1B1C         move.l  aØ,(aØ)            ; Header initialisieren
1B1E         addq.l  #4,(aØ)
1B2Ø         clr.l   4(aØ)
1B24         move.l  aØ,8(aØ)
1B28         lea     188(a6),aØ         ; aØ -> Port List Header
1B2C         bra     FC1682             ; ---> Forbid, Enqueue, Permit
1B3Ø
1B3Ø ;---------------------------------  RemPort
1B3Ø
1B3Ø FC1B3Ø  bra     FC168E             ; ---> Forbid, Remove, Permit
1B34
1B34 ;---------------------------------  PutMsg
1B34
1B34 FC1B34  move.b  #5,8(a1)           ; Typ := 'Message'
1B3A FC1B3A  move.l  aØ,d1              ; d1 := aØ -> Port
1B3C         lea     14(aØ),aØ          ; aØ -> Message List Header im Port
1B4Ø         move.w  #4ØØØ,INTENA       ; Interrupts sperren
1B48         addq.b  #1,126(a6)         ; Disable
1B4C         lea     4(aØ),aØ           ; AddTail
1B5Ø         move.l  4(aØ),dØ
1B54         move.l  a1,4(aØ)
1B58         move.l  aØ,(a1)
1B5A         move.l  dØ,4(a1)
1B5E         movea.l dØ,aØ
1B6Ø         move.l  a1,(aØ)
1B62         movea.l d1,a1              ; a1 -> Port
1B64         move.l  1Ø(a1),d1          ; d1 := SigTask
1B68         beq.s   FC1B9E             ; keine Signale gesetzt: ->
1B6A         move.b  E(a1),dØ           ; dØ := mp_Flags
1B6E         andi.w  #3,dØ              ; Action-Flags isolieren
1B72         beq.s   FC1B8E             ; nicht gesetzt: ->
1B74         cmpi.b  #1,dØ              ; SoftInt?
1B78         bne.s   FC1B82             ; nein: ->
```

```
1B7A         movea.l d1,a1          ; a1 -> SigTask
1B7C         jsr     -B4(a6)        ; ---> Cause SoftInt
1B80         bra.s   FC1B9E         ; --->
1B82
1B82 FC1B82  cmpi.b  #2,d0          ; pa_Ignore?
1B86         beq.s   FC1B9E         ; ja: ->
1B88         movea.l d1,a0          ; a0 -> SigTask
1B8A         jsr     (a0)           ; ---> Task aufrufen
1B8C         bra.s   FC1B9E         ; --->
1B8E
1B8E FC1B8E  move.b  F(a1),d0       ; d0 := SigBit
1B92         movea.l d1,a1          ; a1 -> Port
1B94         moveq   #0,d1          ; d1.l := 0
1B96         bset.l  d0,d1          ; SigBit in d1 setzen
1B98         move.l  d1,d0          ; d0 := d1
1B9A         jsr     -144(a6)       ; ---> Signal
1B9E FC1B9E  subq.b  #1,126(a6)     ; Enable
1BA2         bge.s   FC1BAC         ; Noch keine Freigabe: ->
1BA4         move.w  #C000,INTENA   ; Interrupts freigeben
1BAC FC1BAC  rts
1BAE
1BAE ;--------------------------------  GetMsg
1BAE
1BAE FC1BAE  lea     14(a0),a0      ; a0 -> mp_MsgList
1BB2         move.w  #4000,INTENA   ; Interrupts sperren
1BBA         addq.b  #1,126(a6)     ; Disable
1BBE         movea.l (a0),a1        ; RemHead
1BC0         move.l  (a1),d0
1BC2         beq.s   FC1BCC
1BC4         move.l  d0,(a0)
1BC6         exg     d0,a1
1BC8         move.l  a0,4(a1)
1BCC FC1BCC  subq.b  #1,126(a6)     ; Enable
1BD0         bge.s   FC1BDA         ; Noch keine Freigabe: ->
1BD2         move.w  #C000,INTENA   ; Interrupts freigeben
1BDA FC1BDA  rts
1BDC
1BDC ;--------------------------------  ReplyMsg
1BDC
1BDC FC1BDC  move.l  E(a1),d0       ; d0 := ReplyPort
```

```
1BE0         bne.s    FC1BEA           ; vorhanden: ->
1BE2         move.b   #6,8(a1)         ; Typ 'FreeMsg' eintragen
1BE8         rts
1BEA
1BEA FC1BEA  move.b   #7,8(a1)         ; Typ 'ReplyMsg' eintragen
1BF0         movea.l  d0,a0            ; a0 -> ReplyPort
1BF2         bra      FC1B3A           ; ---> PutMsg
1BF6
1BF6 ;---------------------------------  WaitPort
1BF6
1BF6 FC1BF6  movea.l  14(a0),a1        ; a1 -> 1. Message
1BFA         tst.l    (a1)             ; Message vorhanden?
1BFC         bne.s    FC1C1A           ; ja: ->
1BFE         move.b   F(a0),d1         ; d1 := mp_SigBit
1C02         lea      14(a0),a0        ; a0 -> mp_SigList
1C06         moveq    #0,d0            ; d0.l := 0
1C08         bset.l   d1,d0            ; SigBit in d0 setzen
1C0A         move.l   a2,-(a7)         ; a2 retten
1C0C         movea.l  a0,a2            ; a2 := a0 -> mp_MsgList
1C0E FC1C0E  jsr      -13E(a6)         ; ---> Wait
1C12         movea.l  (a2),a1          ; a1 -> 1. Message
1C14         tst.l    (a1)             ; Message vorhanden?
1C16         beq.s    FC1C0E           ; nein: warten ->
1C18         movea.l  (a7)+,a2         ; a2 wiederherstellen
1C1A FC1C1A  move.l   a1,d0            ; d0 := a1 -> Message
1C1C         rts
1C1E
1C1E ;---------------------------------  FindPort
1C1E
1C1E FC1C1E  lea      188(a6),a0       ; a0 -> Port List Header
1C22         jsr      -114(a6)         ; ---> FindName
1C26         rts
1C28
1C28 ;---------------------------------  AddResource
1C28
1C28 FC1C28  lea      150(a6),a0       ; a0 -> Resource List Header
1C2C         bra      FC1682           ; ---> Forbid, Enqueue, Permit
1C30
1C30 ;---------------------------------  RemResource
1C30
```

```
1C30 FC1C30  bra      FC168E            ; ---> Forbid, Remove, Permit
1C34
1C34 ;--------------------------------  OpenResource
1C34
1C34 FC1C34  lea      150(a6),a0        ; a0 -> Resource List Header
1C38         addq.b   #1,127(a6)        ; Forbid
1C3C         bsr      FC165A            ; ---> FindName
1C40         jsr      -8A(a6)           ; ---> Permit
1C44         rts
1C46
1C46         DC.W     0
1C48
1C48 ;--------------------------------  AddTask
1C48
1C48 FC1C48  moveq    #0,d1             ; d1 := 0
1C4A         move.b   #1,F(a1)          ; tc_State := 1 = 'ts_added'
1C50         move.b   d1,E(a1)          ; tc_Flags := 0
1C54         move.w   #FFFF,10(a1)      ; id_NestCnt, td_NestCnt rücksetzen
1C5A         move.l   13C(a6),12(a1)    ; tc_SigAlloc vorbelegen
1C60         move.l   d1,16(a1)         ; tc_SigWait := 0
1C64         move.l   d1,1A(a1)         ; tc_SigRecvd := 0
1C68         move.l   d1,1E(a1)         ; tc_SigExcept := 0
1C6C         move.w   140(a6),22(a1)    ; tc_TrapAlloc vorbelegen
1C72         move.w   d1,24(a1)         ; tc_Trapable := 0
1C76         tst.l    32(a1)            ; tc_Trapcode
1C7A         bne.s    FC1C82            ; vorhanden: ->
1C7C         move.l   130(a6),32(a1)    ; TaskTrapCode als Default eintragen
1C82 FC1C82  tst.l    2A(a1)            ; tc_ExceptCode
1C86         bne.s    FC1C8E            ; vorhanden: ->
1C88         move.l   134(a6),2A(a1)    ; TaskExceptCode als Default eintragen
1C8E FC1C8E  movea.l  36(a1),a0         ; a0 := tc_SPReg
1C92         move.l   a3,-(a0)          ; FinalPC auf TaskStack
1C94         bne.s    FC1C9A            ; vorhanden: ->
1C96         move.l   138(a6),(a0)      ; durch TaskExitCode ersetzen
1C9A FC1C9A  moveq    #E,d1             ; d1 := 14 = Zählerwert
1C9C FC1C9C  clr.l    -(a0)             ; 15 Langworte auf TaskStack löschen
1C9E         dbra     d1,FC1C9C         ; als Register-Anfangswerte
1CA2         clr.w    -(a0)             ; 1 weiteres Wort für Statusregister
1CA4         move.l   a2,-(a0)          ; PC-Anfangswert auf TaskStack
1CA6         btst.b   #4,129(a6)        ; M 68881 Coprozessor im System?
```

```
1CAC         beq.s    FC1CB2          ; nein: ->
1CAE         moveq    #0,d0           ; d0 := 0
1CB0         move.l   d0,-(a0)        ; auf TaskStack
1CB2 FC1CB2  move.l   a0,36(a1)       ; tc_SPReg setzen
1CB6         lea      196(a6),a0      ; a0 -> Task Ready List Header
1CBA         move.w   #4000,INTENA    ; Interrupts sperren
1CC2         addq.b   #1,126(a6)      ; Disable
1CC6         move.b   #3,F(a1)        ; tc_State := 3 = 'Ready'
1CCC         bsr      FC1634          ; ---> Enqueue
1CD0         move.l   196(a6),d0      ; d0 := lh_Head
1CD4         subq.b   #1,126(a6)      ; Enable
1CD8         bge.s    FC1CE2          ; Noch keine Freigabe: ->
1CDA         move.w   #C000,INTENA    ; Interrupts freigeben
1CE2 FC1CE2  cmp.l    a1,d0           ; Steht Task am Anfang der Liste?
1CE4         bne.s    FC1CEA          ; nein: ->
1CE6         jsr      -30(a6)         ; ---> Reschedule
1CEA FC1CEA  rts
1CEC
1CEC ;------ Task-Abschluß
1CEC
1CEC FC1CEC  movea.l 4,a6             ; a6 := SysBase
1CF0         moveq    #0,d0           ; d0 := 0
1CF2         movea.l d0,a1            ; a1 := d0 = 0: Flag 'Laufende Task'
1CF4
1CF4 ;---------------------------------  RemTask
1CF4
1CF4 FC1CF4  movem.l d2-d3,-(a7)      ; Register retten
1CF8         move.l   a1,d3           ; d3 -> Task
1CFA         bne.s    FC1D02          ; nicht die laufende Task: ->
1CFC         move.l   114(a6),d3      ; d3 := ThisTask
1D00         bra.s    FC1D26          ; --->
1D02
1D02 FC1D02  cmpa.l   114(a6),a1      ; a1 = ThisTask?
1D06         beq.s    FC1D26          ; ja: ->
1D08         move.w   #4000,INTENA    ; Interrupts sperren
1D10         addq.b   #1,126(a6)      ; Disable
1D14         bsr      FC1600          ; ---> Remove
1D18         subq.b   #1,126(a6)      ; Enable
1D1C         bge.s    FC1D26          ; Noch keine Freigabe: ->
1D1E         move.w   #C000,INTENA    ; Interrupts freigeben
```

```
1D26 FC1D26  movea.l d3,a1            ; a1 := d3 -> Task
1D28         move.b  #6,F(a1)         ; tc_State := 6 = 'Removed'
1D2E         cmpa.l  114(a6),a1       ; a1 = ThisTask?
1D32         bne.s   FC1D38           ; nein: ->
1D34         addq.b  #1,127(a6)       ; Forbid
1D38 FC1D38  lea     4A(a1),a0        ; a0 -> MemEntry List Header
1D3C         move.l  (a0),d2          ; d2 := lh_Head
1D3E         beq.s   FC1D58           ; Kein Eintrag im Header: ->
1D40         cmpa.l  8(a0),a0         ; Liste leer?
1D44         beq     FC1D58           ; ja: ->
1D48         clr.l   (a0)             ; lh_Head löschen
1D4A FC1D4A  movea.l d2,a0            ; a0 -> MemList
1D4C         move.l  (a0),d2          ; d2 := ln_Succ
1D4E         beq     FC1D58           ; Liste zu Ende: ->
1D52         jsr     -E4(a6)          ; ---> FreeEntry
1D56         bra.s   FC1D4A           ; ---> Loop
1D58
1D58 FC1D58  cmp.l   114(a6),d3       ; d3 = ThisTask?
1D5C         bne.s   FC1D6C           ; nein: ->
1D5E         lea     6(pc),a5         ; a5 := FC1D66: Rückkehradresse
1D62         jmp     -1E(a6)          ; ---> Supervisor
1D66
1D66 FC1D66  addq.l  #6,a7            ; Supervisor-Einträge vom Stack nehmen
1D68         jmp     -3C(a6)          ; ---> Dispatch
1D6C
1D6C FC1D6C  moveq   #0,d0            ; d0 := 0
1D6E         movem.l (a7)+,d2-d3      ; Register wiederherstellen
1D72         rts
1D74
1D74 ;--------------------------------  FindTask
1D74
1D74 FC1D74  move.l  a1,d0            ; d0 := a1
1D76         bne.s   FC1D7E           ; Nicht die laufende Task: ->
1D78         move.l  114(a6),d0       ; d0 := ThisTask
1D7C         bra.s   FC1DC6           ; --->
1D7E
1D7E FC1D7E  lea     196(a6),a0       ; a0 -> Ready List Header
1D82         move.w  #4000,INTENA     ; Interrupts sperren
1D8A         addq.b  #1,126(a6)       ; Disable
1D8E         jsr     -114(a6)         ; ---> FindName
```

```
1D92          tst.l   d0               ; gefunden?
1D94          bne.s   FC1DB8           ; ja: ->
1D96          lea     1A4(a6),a0       ; a0 -> Wait List Header
1D9A          jsr     -114(a6)         ; ---> FindName
1D9E          tst.l   d0               ; gefunden?
1DA0          bne.s   FC1DB8           ; ja: ->
1DA2          movea.l 114(a6),a0       ; a0 := ThisTask
1DA6          movea.l A(a0),a0         ; a0 -> Name der laufenden Task
1DAA FC1DAA   cmpm.b  (a0)+,(a1)+      ; zeichenweiser Namensvergleich
1DAC          bne.s   FC1DB8           ; ungleich: ->
1DAE          tst.b   -1(a0)           ; Endemarke erreicht?
1DB2          bne.s   FC1DAA           ; nein: Vergleich fortsetzen ->
1DB4          move.l  114(a6),d0       ; d0 := ThisTask
1DB8 FC1DB8   subq.b  #1,126(a6)       ; Enable
1DBC          bge.s   FC1DC6           ; Noch keine Freigabe: ->
1DBE          move.w  #C000,INTENA     ; Interrupts freigeben
1DC6 FC1DC6   rts
1DC8
1DC8 ;---------------------------------  SetTaskPri
1DC8
1DC8 FC1DC8   move.w  #4000,INTENA     ; Interrupts sperren
1DD0          addq.b  #1,126(a6)       ; Disable
1DD4          move.b  9(a1),-(a7)      ; Alte Priorität auf Stack retten
1DD8          move.b  d0,9(a1)         ; Neue Priorität eintragen
1DDC          cmpa.l  114(a6),a1       ; a1 = ThisTask?
1DE0          beq.s   FC1E00           ; ja: ->
1DE2          cmpi.b  #3,F(a1)         ; tc_State = 'Ready'?
1DE8          bne.s   FC1E04           ; nein: ->
1DEA          move.l  a1,d0            ; a1 in d0 retten
1DEC          bsr     FC1600           ; ---> Remove Task aus Ready List
1DF0          lea     196(a6),a0       ; a0 -> Ready List Header
1DF4          movea.l d0,a1            ; a1 wiederherstellen
1DF6          bsr     FC1634           ; ---> Enqueue mit neuer Priorität
1DFA          cmpa.l  196(a6),a1       ; Steht Task am Anfang der Liste?
1DFE          bne.s   FC1E04           ; nein: ->
1E00 FC1E00   jsr     -30(a6)          ; ---> Reschedule
1E04 FC1E04   subq.b  #1,126(a6)       ; Enable
1E08          bge.s   FC1E12           ; Noch keine Freigabe: ->
1E0A          move.w  #C000,INTENA     ; Interrupts freigeben
1E12 FC1E12   moveq   #0,d0            ; d0.l := 0
```

```
1E14        move.b  (a7)+,dØ          ; dØ := alte Priorität
1E16        rts
1E18
1E18 ;--------------------------------  SetExcept
1E18
1E18 FC1E18 movea.l 114(a6),a1        ; a1 := ThisTask
1E1C        lea     1E(a1),aØ         ; aØ -> tc_SigExcept
1E2Ø        bra.s   FC1E2A            ; --->
1E22
1E22 ;--------------------------------  SetSignal
1E22
1E22 FC1E22 movea.l 114(a6),a1        ; a1 := ThisTask
1E26        lea     1A(a1),aØ         ; aØ -> tc_SigRecvd
1E2A FC1E2A and.l   d1,dØ             ; Neue Signale maskieren
1E2C        move.w  #4ØØØ,INTENA      ; Interrupts sperren
1E34        addq.b  #1,126(a6)        ; Disable
1E38        move.l  (aØ),-(a7)        ; Alte Signale auf Stack retten
1E3A        not.l   d1                ; Signalmaske invertieren
1E3C        and.l   (aØ),d1           ; Signale neu setzen
1E3E        or.l    dØ,d1
1E4Ø        move.l  d1,(aØ)
1E42        move.l  1A(a1),dØ         ; dØ := tc_SigRecvd
1E46        bra.s   FC1E5C            ; --->
1E48
1E48 ;--------------------------------  Signal
1E48
1E48 FC1E48 lea     1A(a1),aØ         ; aØ -> tc_SigRecvd
1E4C        move.w  #4ØØØ,INTENA      ; Interrupts sperren
1E54        addq.b  #1,126(a6)        ; Disable
1E58        move.l  (aØ),-(a7)        ; Alte Signale auf Stack retten
1E5A        or.l    dØ,(aØ)           ; Neues Signal einodern
1E5C FC1E5C move.l  1E(a1),d1         ; d1 := tc_SigExcept
1E6Ø        and.l   dØ,d1             ; mit neuem Signal odern
1E62        bne.s   FC1EAE            ; Neues Signal ist Except-Signal: ->
1E64        cmpi.b  #4,F(a1)          ; Status 'Wait'?
1E6A        bne.s   FC1EBE            ; nein: ->
1E6C        and.l   16(a1),dØ         ; Wartet Task auf dieses Signal?
1E7Ø        beq.s   FC1EBE            ; nein: ->
1E72 FC1E72 lea     1A4(a6),aØ        ; aØ -> Wait List Header
1E76        move.l  a1,dØ             ; Remove Task aus Wait List
```

```
1E78          movea.l (a1),a0
1E7A          movea.l 4(a1),a1
1E7E          move.l  a0,(a1)
1E80          move.l  a1,4(a0)
1E84          movea.l d0,a1               ; a1 := d0 -> Task
1E86          move.b  #3,F(a1)            ; tc_State := 'Ready'
1E8C          lea     196(a6),a0          ; a0 -> Ready List Header
1E90          bsr     FC1634              ; ---> Enqueue Task in Ready List
1E94          cmpa.l  196(a6),a1          ; Steht Task am Anfang der Liste?
1E98          bne.s   FC1EBE              ; nein: ->
1E9A FC1E9A   subq.b  #1,126(a6)          ; Enable
1E9E          bge.s   FC1EA8              ; Noch keine Freigabe: ->
1EA0          move.w  #C000,INTENA        ; Interrupts freigeben
1EA8 FC1EA8   move.l  (a7)+,d0            ; d0 := alte Signale
1EAA          jmp     -30(a6)             ; ---> Reschedule
1EAE
1EAE FC1EAE   bset.b  #5,E(a1)            ; 'Except' in tc_Flags setzen
1EB4          cmpi.b  #4,F(a1)            ; tc_State = 'Wait'?
1EBA          beq.s   FC1E72              ; ja: Task in Ready List setzen ->
1EBC          bra.s   FC1E9A              ; sonst ->
1EBE
1EBE FC1EBE   subq.b  #1,126(a6)          ; Enable
1EC2          bge.s   FC1ECC              ; Noch keine Freigabe: ->
1EC4          move.w  #C000,INTENA        ; Interrupts freigeben
1ECC FC1ECC   move.l  (a7)+,d0            ; d0 := alte Signale
1ECE          rts
1ED0
1ED0 ;---------------------------------   Wait
1ED0
1ED0 FC1ED0   movea.l 114(a6),a1          ; a1 := ThisTask
1ED4          move.l  d0,16(a1)           ; tc_SigWait := d0
1ED8          move.w  #4000,INTENA        ; Interrupts sperren
1EE0          addq.b  #1,126(a6)          ; Disable
1EE4          bra.s   FC1F1A              ; --->
1EE6
1EE6 FC1EE6   move.b  #4,F(a1)            ; tc_State := 4 = 'Wait'
1EEC          lea     1A4(a6),a0          ; a0 -> Task Wait List
1EF0          lea     4(a0),a0            ; AddTail: Task am Listenende anfügen
1EF4          move.l  4(a0),d0
1EF8          move.l  a1,4(a0)
```

```
1EFC        move.l  a0,(a1)
1EFE        move.l  d0,4(a1)
1F02        movea.l d0,a0
1F04        move.l  a1,(a0)
1F06        movea.l a5,a0                 ; a5 in a0 retten
1F08        lea     -36(a6),a5            ; a5 -> Switch
1F0C        jsr     -1E(a6)               ; ---> Supervisor, Switch
1F10        movea.l a0,a5                 ; a5 wiederherstellen
1F12        movea.l 114(a6),a1            ; a1 := ThisTask
1F16        move.l  16(a1),d0             ; d0 := tc_SigWait
1F1A FC1F1A move.l  1A(a1),d1             ; d1 := tc_SigRecvd
1F1E        and.l   d0,d1                 ; Erwartetes Signal empfangen?
1F20        beq.s   FC1EE6                ; nein: warten ->
1F22        eor.l   d1,1A(a1)             ; Signalbit löschen
1F26        subq.b  #1,126(a6)            ; Enable
1F2A        bge.s   FC1F34                ; Noch keine Freigabe: ->
1F2C        move.w  #C000,INTENA          ; Interrupts freigeben
1F34 FC1F34 move.l  d1,d0                 ; d0 := Empfangenes Signal
1F36        rts
1F38
1F38 ;--------------------------------  Reschedule
1F38
1F38 FC1F38 bset.b  #7,124(a6)            ; SAR = 'Scheduling Attn reqd' setzen
1F3E        sne     d0                    ; d0 := -1, wenn Flag gesetzt war
1F40        tst.b   127(a6)               ; Forbid gesetzt?
1F44        bge.s   FC1F58                ; ja: ->
1F46        tst.b   126(a6)               ; Disable gesetzt?
1F4A        blt.s   FC1F74                ; nein: ->
1F4C        tst.b   d0                    ; war SAR gesetzt?
1F4E        bne.s   FC1F58                ; ja: ->
1F50        move.w  #8004,INTREQ          ; Request SoftInt
1F58 FC1F58 rts
1F5A
1F5A ;--------------------------------  Forbid
1F5A
1F5A FC1F5A addq.b  #1,127(a6)            ; TDNestCnt erhöhen
1F5E        rts
1F60
1F60 ;--------------------------------  Permit
1F60
```

```
1F60 FC1F60  subq.b  #1,127(a6)          ; TDNestCnt erniedrigen
1F64         bge.s   FC1F80              ; Noch nicht -1: ->
1F66         tst.b   126(a6)             ; Disable gesetzt?
1F6A         bge.s   FC1F80              ; ja: ->
1F6C         btst.b  #7,124(a6)          ; SAR gesetzt?
1F72         beq.s   FC1F80              ; nein: ->
1F74 FC1F74  move.l  a5,-(a7)            ; a5 auf Stack retten
1F76         lea     A(pc),a5            ; a5 := FC1F82: Rückkehradresse
1F7A         jsr     -1E(a6)             ; ---> Supervisor
1F7E         movea.l (a7)+,a5            ; a5 wiederherstellen
1F80 FC1F80  rts
1F82
1F82 FC1F82  btst.b  #5,(a7)             ; War Supervisor-Mode schon gesetzt?
1F86         beq.s   FC1F8A              ; nein: ->
1F88         rte
1F8A
1F8A FC1F8A  jmp     -2A(a6)             ; ---> Schedule
1F8E
1F8E ;---------------------------------   AllocTrap
1F8E
1F8E FC1F8E  movea.l 114(a6),a1          ; a1 := ThisTask
1F92         move.w  22(a1),d1           ; d1 := tc_TrapAlloc
1F96         cmpi.b  #FF,d0              ; spezielle Trapnummer angefordert?
1F9A         beq.s   FC1FA2              ; nein: ->
1F9C         bset.l  d0,d1               ; Trapbit in tc_TrapAlloc setzen
1F9E         beq.s   FC1FAE              ; war noch nicht gesetzt: ->
1FA0         bra.s   FC1FAC              ; sonst Fehler ->
1FA2
1FA2 FC1FA2  moveq   #F,d0               ; d0 := höchste mögliche Trapnummer
1FA4 FC1FA4  bset.l  d0,d1               ; in tc_TrapAlloc setzen
1FA6         beq.s   FC1FAE              ; war noch frei: ->
1FA8         dbra    d0,FC1FA4           ; sonst nächste Nummer versuchen ->
1FAC FC1FAC  moveq   #FF,d0              ; Kein Trap frei: d0 := -1
1FAE FC1FAE  move.w  d1,22(a1)           ; tc_TrapAlloc zurückschreiben
1FB2         rts
1FB4
1FB4 ;---------------------------------   FreeTrap
1FB4
1FB4 FC1FB4  movea.l 114(a6),a1          ; a1 := ThisTask
1FB8         move.w  22(a1),d1           ; d1 := tc_TrapAlloc
```

```
1FBC         bclr.l  d0,d1          ; Trap freigeben
1FBE         move.w  d1,22(a1)      ; tc_TrapAlloc zurückschreiben
1FC2         rts
1FC4
1FC4 ;--------------------------------  AllocSignal
1FC4
1FC4 FC1FC4  movea.l 114(a6),a1     ; a1 := ThisTask
1FC8         move.l  12(a1),d1      ; d1 := tc_SigAlloc
1FCC         cmpi.b  #FF,d0         ; spezielle Signalnummer angefordert?
1FD0         beq.s   FC1FD8         ; nein: ->
1FD2         bset.l  d0,d1          ; Signalbit setzen
1FD4         beq.s   FC1FE6         ; war noch frei: ->
1FD6         bra.s   FC1FE2         ; sonst Fehler ->
1FD8
1FD8 FC1FD8  moveq   #1F,d0         ; d0 := höchste mögliche Signalnummer
1FDA FC1FDA  bset.l  d0,d1          ; Signalbit setzen
1FDC         beq.s   FC1FE6         ; war noch frei: ->
1FDE         dbra    d0,FC1FDA      ; sonst nächste Signalnr versuchen ->
1FE2 FC1FE2  moveq   #FF,d0         ; Kein Signal frei: d0 := -1
1FE4         bra.s   FC1FFA         ; --->
1FE6
1FE6 FC1FE6  move.l  d1,12(a1)      ; Signale zurückschreiben
1FEA         moveq   #FF,d1         ; Alle Bits in d1 setzen
1FEC         bclr.l  d0,d1          ; Neues Signalbit löschen
1FEE         and.l   d1,1A(a1)      ; in tc_SigRecvd,
1FF2         and.l   d1,1E(a1)      ; in tc_SigExcept und
1FF6         and.l   d1,16(a1)      ; in tc_SigWait
1FFA FC1FFA  rts
1FFC
1FFC ;--------------------------------  FreeSignal
1FFC
1FFC FC1FFC  movea.l 114(a6),a1     ; a1 := ThisTask
2000         move.l  12(a1),d1      ; d1 := tc_SigAlloc
2004         bclr.l  d0,d1          ; Signal löschen
2006         move.l  d1,12(a1)      ; tc_SigAlloc zurückschreiben
200A         rts
200C
200C ;--------------------------------  RawDoFmt - Subroutinen
200C
200C ;------ Stringlänge ermitteln
```

```
200C
200C FC200C  moveq   #FF,d2            ; d2 := -1
200E FC200E  tst.b   (a0)+             ; Zeigt a0 auf Endemarke?
2010         dbeq    d2,FC200E         ; nein: d2 vermindern, ->
2014         neg.l   d2                ; d2 negieren
2016         subq.w  #1,d2             ; und 1 subtrahieren
2018         rts
201A
201A ;------ Dezimal-String in Binärzahl wandeln
201A
201A FC201A  clr.l   d0                ; d0 := 0  Ergebnisregister
201C         clr.l   d2                ; d2 := 0  Nächste Ziffer
201E FC201E  move.b  (a4)+,d2          ; d2 := Zeichen aus String
2020         cmpi.b  #'0',d2           ; Dezimalziffer?
2024         bcs.s   FC203E
2026         cmpi.b  #'9',d2
202A         bhi.s   FC203E            ; nein: ->
202C         add.l   d0,d0             ; Ergebnisregister
202E         move.l  d0,d1             ; mit 10 multiplizieren
2030         add.l   d0,d0
2032         add.l   d0,d0
2034         add.l   d1,d0
2036         subi.b  #'0',d2           ; d2 := Wert der Ziffer
203A         add.l   d2,d0             ; zum Ergebnisregister addieren
203C         bra.s   FC201E            ; ---> Loop
203E
203E FC203E  subq.l  #1,a4             ; Zeiger in String zurücksetzen
2040         rts
2042
2042 ;------ Binärzahl in Dezimal-String wandeln
2042
2042 FC2042  tst.l   d4                ; Zahl in d4
2044         beq.s   FC207A            ; = 0: ->
2046         bmi.s   FC204C            ; negativ: ->
2048         neg.l   d4                ; Zahl negieren
204A         bra.s   FC2050            ; --->
204C
204C FC204C  move.b  #'-',(a5)+        ; Minuszeichen in Puffer
2050 FC2050  lea     FC2084,a0         ; a0 -> Tabelle der Zehnerpotenzen
2056         clr.w   d1                ; d1 := 0 (Flag für Anfangsnullen)
```

```
2058 FC2058 move.l  (a0)+,d2        ; d2 := nächster Wert aus Tabelle
205A        beq.s   FC207A          ; Tabellenende erreicht: ->
205C        moveq   #FF,d0          ; d0 := -1
205E FC205E add.l   d2,d4           ; so oft d2 zur Zahl addieren,
2060        dbgt    d0,FC205E       ; bis Ergebnis positiv ist,
2064        sub.l   d2,d4           ; dann einmal subtrahieren
2066        addq.w  #1,d0           ; wurde nur einmal addiert?
2068        bne.s   FC206E          ; nein: ->
206A        tst.w   d1              ; liegt eine 'führende Null' vor?
206C        beq.s   FC2058          ; ja: nicht ausgeben ->
206E FC206E moveq   #FF,d1          ; Flag für Anfangsnullen setzen
2070        neg.b   d0              ; d0 := Zahl der Additionen - 1
2072        addi.b  #'0',d0         ; ergibt Ziffernkode
2076        move.b  d0,(a5)+        ; Ziffer in Puffer
2078        bra.s   FC2058          ; ---> Loop
207A
207A FC207A neg.b   d4              ; d4 := Wert der letzten Ziffer
207C        addi.b  #'0',d4         ; ergibt Ziffernkode
2080        move.b  d4,(a5)+        ; Ziffer in Puffer
2082        rts
2084
2084 ;------ Tabelle der Zehnerpotenzen
2084
2084 FC2084 DC.L    3B9ACA00        ; 1 000 000 000
2088        DC.L    5F5E100         ;   100 000 000
208C        DC.L    989680          ;    10 000 000
2090        DC.L    F4240           ;     1 000 000
2094        DC.L    186A0           ;       100 000
2098        DC.L    2710            ;        10 000
209C        DC.L    3E8             ;         1 000
20A0        DC.L    64              ;           100
20A4        DC.L    A               ;            10
20A8        DC.L    0               ; Endemarke
20AC
20AC ;------ Binärzahl in Hex-String wandeln
20AC
20AC FC20AC tst.l   d4              ; Zahl in d4
20AE        beq.s   FC207A          ; = 0: Ziffer '0' ausgeben ->
20B0        clr.w   d1              ; d1 := 0 = Flag für Anfangsnullen
20B2        btst.l  #2,d3           ; Bit 2 in d3 gesetzt?
```

```
20B6          bne.s    FC20BE           ; ja: 8 Stellen ->
20B8          moveq    #3,d2            ; d2 := 3 ergibt 4 Stellen
20BA          swap     d4               ; Zahlwort in oberen Teil holen
20BC          bra.s    FC20C0           ; --->
20BE
20BE FC20BE   moveq    #7,d2            ; d2 := 7 ergibt 8 Stellen
20C0 FC20C0   rol.l    #4,d4            ; d4, Bits 0 bis 3 := nächste Stelle
20C2          move.b   d4,d0            ; d0.b := d4.b
20C4          andi.b   #F,d0            ; Stelle isolieren
20C8          bne.s    FC20CE           ; nicht Null: ->
20CA          tst.w    d1               ; Anfangsnull?
20CC          beq.s    FC20E2           ; ja: nicht ausgeben ->
20CE FC20CE   moveq    #FF,d1           ; Flag für Anfangsnullen setzen
20D0          cmpi.b   #9,d0            ; Stellenwert > 9?
20D4          bhi.s    FC20DC           ; ja: ->
20D6          addi.b   #'0',d0          ; ergibt Ziffernkode
20DA          bra.s    FC20E0           ; --->
20DC
20DC FC20DC   addi.b   #37,d0           ; ergibt Buchstabenkode 'a',...,'f'
20E0 FC20E0   move.b   d0,(a5)+         ; Ziffer in Puffer
20E2 FC20E2   dbra     d2,FC20C0        ; nächste Ziffer ermitteln ->
20E6          rts
20E8
20E8 ;--------------------------------  RawDoFmt
20E8
20E8 FC20E8   movem.l d2-d6/a2-a5,-(a7)  ; Register retten
20EC          link     a6,#-10          ; Ausgabepuffer einrichten
20F0          move.l   a1,-(a7)         ; a1 -> Ausgabedaten: auf Stack retten
20F2          movea.l  a0,a4            ; a4 := a0 -> Formatstring
20F4 FC20F4   move.b   (a4)+,d0         ; d0 := Zeichen aus Formatstring
20F6          beq.s    FC2102           ; Endemarke: ->
20F8          cmpi.b   #'%',d0          ; folgt Formatanweisung?
20FC          beq.s    FC210C           ; ja: ->
20FE FC20FE   jsr      (a2)             ; ---> Zeichen direkt ausgeben
2100          bra.s    FC20F4           ; ---> Loop
2102
2102 FC2102   jsr      (a2)             ; ---> Zeichen direkt ausgeben
2104          unlk     a6               ; Ausgabepuffer auflösen
2106          movem.l (a7)+,d2-d6/a2-a5  ; Register wiederherstellen
210A          rts
```

```
210C
210C ;------ Formatanweisung bearbeiten
210C
210C FC210C lea     -10(a6),a5       ; a5 -> Ausgabepuffer
2110        clr.w   d3               ; d3.w := 0  (Flags)
2112        cmpi.b  #'-',(a4)        ; Erstes Zeichen '-'?
2116        bne.s   FC211E           ; nein: ->
2118        bset.l  #0,d3            ; Flagbit 0: Ausgabe linksbündig
211C        addq.l  #1,a4            ; Formatzeiger vorrücken
211E FC211E cmpi.b  #'0',(a4)        ; Nächstes Zeichen '0'?
2122        bne.s   FC2128           ; nein: ->
2124        bset.l  #1,d3            ; Flagbit 1: mit Nullen auffüllen
2128 FC2128 bsr     FC201A           ; ---> Dezimalstring-Binär-Wandlung
212C        move.w  d0,d6            ; d6 := d0 = minimale Feldbreite
212E        clr.l   d5               ; d5 := 0
2130        cmpi.b  #'.',(a4)        ; Nächstes Zeichen '.'?
2134        bne.s   FC213E           ; nein: ->
2136        addq.w  #1,a4            ; Formatzeiger vorrücken
2138        bsr     FC201A           ; ---> Dezimalstring-Binär-Wandlung
213C        move.w  d0,d5            ; d5 := d0 = maximale Feldbreite
213E FC213E cmpi.b  #'l',(a4)        ; Nächstes Zeichen 'l'?
2142        bne.s   FC214A           ; nein: ->
2144        bset.l  #2,d3            ; Flagbit 2: Argument 'long'
2148        addq.w  #1,a4            ; Formatzeiger vorrücken
214A FC214A move.b  (a4)+,d0         ; d0 := nächstes Zeichen
214C        cmpi.b  #'d',d0          ; 'd': Dezimaldarstellung?
2150        bne.s   FC215A           ; nein: ->
2152        bsr.s   FC2168           ; ---> Ausgabedaten nach d4 holen
2154        bsr     FC2042           ; ---> Binär-Dezimalstring-Wandlung
2158        bra.s   FC21A2           ; ---> Abschluß
215A
215A FC215A cmpi.b  #'x',d0          ; 'x': Hex-Darstellung?
215E        bne.s   FC2188           ; nein: ->
2160        bsr.s   FC2168           ; ---> Ausgabedaten nach d4 holen
2162        bsr     FC20AC           ; ---> Binär-Hexstring-Wandlung
2166        bra.s   FC21A2           ; ---> Abschluß
2168
2168 ;------ Ausgabezahl nach d4 holen
2168
2168 FC2168 btst.l  #2,d3            ; War 'l' angegeben?
```

```
216C         bne.s    FC217C          ; ja: ->
216E         movea.l 4(a7),a1         ; a1 -> Ausgabedaten
2172         move.w   (a1)+,d4        ; d4 := Datenwort
2174         move.l   a1,4(a7)        ; neues a1 zurückschreiben
2178         ext.l    d4              ; Datenwort auf Langwort erweitern
217A         rts
217C
217C FC217C  movea.l 4(a7),a1         ; a1 -> Ausgabedaten
2180         move.l   (a1)+,d4        ; d4 := Datenlangwort
2182         move.l   a1,4(a7)        ; neues a1 zurückschreiben
2186         rts
2188
2188 ;------ Fortsetzung der Formatstring-Bearbeitung
2188
2188 FC2188  cmpi.b  #'s',d0          ; 's': Zeichenkette?
218C         bne.s    FC2196          ; nein: ->
218E         movea.l (a7),a1          ; a1 -> Ausgabedaten
2190         movea.l (a1)+,a5         ; a5 -> Zeichenkette
2192         move.l   a1,(a7)         ; neues a1 zurückspeichern
2194         bra.s    FC21A8          ; ---> Abschluß
2196
2196 FC2196  cmpi.b  #'c',d0          ; 'c': Einzelnes Zeichen?
219A         bne      FC20FE          ; nein: ->
219E         bsr.s    FC2168          ; ---> Zeichenkode nach d4 holen
21A0         move.b   d4,(a5)+        ; und in Puffer schreiben
21A2 FC21A2  clr.b    (a5)            ; Puffer mit Endekode 0 abschließen
21A4         lea      -10(a6),a5      ; a5 -> Pufferanfang
21A8 FC21A8  movea.l a5,a0            ; a0 := a5
21AA         bsr      FC200C          ; ---> Stringlänge im Puffer ermitteln
21AE         tst.w    d5              ; Maximale Feldlänge angegeben?
21B0         beq.s    FC21B6          ; nein: ->
21B2         cmp.w    d5,d2           ; Stringlänge > maximale Feldlänge?
21B4         bhi.s    FC21B8          ; ja: ->
21B6 FC21B6  move.w   d2,d5           ; Feldlänge := Stringlänge
21B8 FC21B8  sub.w    d5,d6           ; d6 := Feldlänge - Stringlänge
21BA         bpl.s    FC21BE          ; Feldlänge >= Stringlänge: ->
21BC         clr.w    d6              ; d6 := 0
21BE FC21BE  btst.l   #0,d3           ; Linksbündige Ausgabe?
21C2         bne.s    FC21CC          ; ja: ->
21C4         bsr.s    FC21DE          ; ---> mit 0 oder Blanks auffüllen
```

```
21C6        bra.s   FC21CC          ; --->
21C8
21C8 FC21C8 move.b  (a5)+,d0        ; d0 := Zeichen aus Puffer
21CA        jsr     (a2)            ; ---> Zeichen ausgeben
21CC FC21CC dbra    d5,FC21C8       ; bis alle Zeichen ausgegeben ->
21D0        btst.l  #0,d3           ; Linksbündige Ausgabe?
21D4        beq     FC20F4          ; nein: ->
21D8        bsr.s   FC21DE          ; ---> mit 0 oder Blanks auffüllen
21DA        bra     FC20F4          ; --->
21DE
21DE ;------ Nullen oder Leerstellen ausgeben
21DE
21DE FC21DE move.b  #' ',d2         ; d2 := Blank-Kode
21E2        btst.l  #1,d3           ; Auffüllung mit Nullen gefordert?
21E6        beq.s   FC21F2          ; nein: ->
21E8        move.b  #'0',d2         ; d2 := '0'
21EC        bra.s   FC21F2          ; --->
21EE
21EE FC21EE move.b  d2,d0           ; d0 := d2 = auszugebendes Zeichen
21F0        jsr     (a2)            ; ---> Zeichen ausgeben
21F2 FC21F2 dbra    d6,FC21EE       ; wiederholen bis d6 = -1 ->
21F6        rts
21F8
21F8 ;---------------------------------  RawIOInit
21F8
21F8 FC21F8 move.w  #174,SERPER     ; Baudrate auf 9600 setzen
2200        rts
2202
2202 ;--------------------------------- RawMayGetChar
2202
2202 FC2202 moveq   #-1,d0          ; Flag 'Kein Zeichen empfangen'
2204        move.w  SERDATR,d1      ; Serielles Datenregister lesen
220A        btst.l  #E,d1           ; Empfangspuffer voll?
220E        beq.s   FC2220          ; nein: kein Zeichen gelesen ->
2210        move.w  #800,INTREQ     ; RBF-Interrupt-Request setzen
2218        andi.l  #7F,d1          ; Bits 0 bis 6 isolieren
221E        move.l  d1,d0           ; d0 := gelesenes Zeichen oder -1
2220 FC2220 rts
2222
2222 ;------ Auf Zeichenempfang warten
```

```
2222
2222 FC2222  bsr.s    FC2202           ; ---> RawMayGetChar
2224         tst.l    d0               ; Zeichen empfangen?
2226         bmi.s    FC2222           ; nein: warten ->
2228         rts
222A
222A ;--------------------------------  RawPutChar
222A
222A         move.l   4(a7),d0         ; d0 := Zeichenkode vom Stack
222E FC222E  tst.b    d0               ; d0 = 0?
2230         beq.s    FC2272           ; ja: fertig ->
2232         move.w   d0,-(a7)         ; d0 auf Stack retten
2234         cmpi.b   #A,d0            ; 'Line Feed'?
2238         bne.s    FC223E           ; nein: ->
223A         moveq    #D,d0            ; zunächst 'Carriage Return'
223C         bsr.s    FC2240           ; ---> senden
223E FC223E  move.w   (a7)+,d0         ; d0 := Zeichen vom Stack
2240 FC2240  move.w   SERDATR,d1       ; Serielles Datenregister lesen
2246         btst.l   #D,d1            ; Sendepuffer leer?
224A         beq.s    FC2240           ; nein: warten ->
224C FC224C  andi.w   #FF,d0           ; Bits 8 bis 15 löschen
2250         ori.w    #100,d0          ; Stopbit setzen
2254         move.w   d0,SERDAT        ; Zeichen in Senderegister schreiben
225A         bsr.s    FC2202           ; ---> RawMayGetChar
225C FC225C  cmpi.b   #13,d0           ; XOFF?
2260         bne.s    FC2266           ; nein: ->
2262         bsr.s    FC2222           ; ---> auf Zeichenempfang warten
2264         bra.s    FC225C           ; ---> Loop
2266
2266 FC2266  cmpi.b   #7F,d0           ; d0 = 127?
226A         bne.s    FC2272           ; nein: ->
226C         bsr      FC232E           ; ---> Debug
2270         bra.s    FC225C           ; ---> Loop
2272
2272 FC2272  rts
2274
2274 ;--------------------------------  PutStr: String ausgeben
2274
2274         movea.l  4(a7),a0         ; a0 -> String
2278 FC2278  move.b   (a0)+,d0         ; d0 := Zeichen aus String
```

```
227A        beq.s    FC228C             ; Endemarke: ->
227C        cmpi.b   #A,dØ              ; 'Line Feed'?
228Ø        bne.s    FC2288             ; nein: ->
2282        moveq    #D,dØ              ; dØ := 'Carriage Return'
2284        bsr.s    FC222E             ; ---> RawPutChar
2286        moveq    #A,dØ              ; dØ := 'Line Feed'
2288 FC2288 bsr.s    FC222E             ; ---> RawPutChar
228A        bra.s    FC2278             ; ---> Loop
228C
228C FC228C rts
228E
228E ;--------------------------------  PutHex: Hex-Zahl ausgeben
228E
228E        movem.l 4(a7),dØ-d1         ; dØ := Wert, d1 := Stellenzahl
2294 FC2294 movem.l d2-d3,-(a7)         ; Register retten
2298        move.l  dØ,d2               ; d2 := dØ = Zahlenwert
229A        moveq   #8,d3               ; d3 := 8
229C        sub.w   d1,d3               ; d3 := 8 - Stellenzahl
229E        bra.s   FC22A2              ; --->
22AØ
22AØ FC22AØ rol.l   #4,d2               ; Zahl in d2 linksbündig machen
22A2 FC22A2 dbra    d3,FC22AØ
22A6        move.w  d1,d3               ; d3 := d1 = Stellenzahl
22A8        bra.s   FC22C2              ; --->
22AA
22AA FC22AA rol.l   #4,d2               ; Nächste Stelle in Byteposition
22AC        moveq   #F,dØ               ; Bits Ø bis 3 setzen
22AE        and.b   d2,dØ               ; dØ := Stellenwert
22BØ        cmpi.b  #9,dØ               ; dØ > 9?
22B4        bls.s   FC22BA              ; nein: ->
22B6        addi.b  #7,dØ               ; zur Erzeugung der Ziffern A,...,F
22BA FC22BA addi.b  #'Ø',dØ             ; ergibt ASCII-Kode
22BE        bsr     FC222E              ; ---> RawPutChar
22C2 FC22C2 dbra    d3,FC22AA           ; bis alle Stellen ausgegeben: ->
22C6        moveq   #' ',dØ             ; Leerstelle
22C8        bsr     FC222E              ; ---> RawPutChar
22CC        movem.l (a7)+,d2-d3         ; Register wiederherstellen
22DØ        rts
22D2
22D2 ;--------------------------------  PutFmt: Formatiert ausgeben
```

```
22D2
22D2 FC22D2  move.l  a2,-(a7)           ; a2 retten
22D4         lea     -A8(pc),a2         ; a2 := FC224C: RawPutChar
22D8         bsr     FC20E8             ; ---> RawDoFmt
22DC         movea.l (a7)+,a2           ; a2 wiederherstellen
22DE         rts
22E0
22E0 ;--------------------------------  Open
22E0
22E0 FC22E0  move.l  a6,d0              ; d0 := LibBase
22E2         addq.w  #1,20(a6)          ; OpenCnt inkrementieren
22E6         rts
22E8
22E8 ;--------------------------------  Close
22E8
22E8 FC22E8  subq.w  #1,20(a6)          ; OpenCnt dekrementieren
22EC
22EC ;--------------------------------  Expunge, Extfunct
22EC
22EC FC22EC  moveq   #0,d0              ; d0 := 0
22EE         rts
22F0
22F0 ;--------------------------------  ROM-Wack
22F0
22F0 FC22F0  DC.B    0A,'rom-wack',0
22FA
22FA ;------ ROMWack initialisieren
22FA
22FA FC22FA  move.l  a6,-(a7)           ; a6 auf Stack retten
22FC         movea.l #200,a6            ; a6 -> ROM-Wack Arbeitsbereich
2302         bsr     FC2472             ; ---> Arbeitsbereich initialisieren
2306         movea.l (a7)+,a6           ; a6 wiederherstellen
2308         move.l  #FC2342,42(a6)     ; DebugEntry setzen
2310         move.l  #FC232E,-70(a6)    ; Debug-Adresse in Sprungliste
2318         bsr     FC21F8             ; ---> RawIOInit
231C         rts
231E
231E ;--------------------------------  Step ausführen
231E
231E FC231E  move.l  #9,-(a7)           ; Nummer des STEP-Vektors auf Stack
```

```
2324        bra.s   FC2342          ; --->
2326
2326 ;--------------------------------  Breakpoint bearbeiten
2326
2326 FC2326 move.l  #2F,-(a7)       ; Nummer des Vektors Trap #15
232C        bra.s   FC2342          ; --->
232E
232E ;--------------------------------  Debug
232E
232E FC232E move.l  a5,-(a7)        ; a5 auf User-Stack retten
2330        lea     6(pc),a5        ; a5 := FC2338 (Rückkehradresse)
2334        jmp     -1E(a6)         ; ---> Supe<rvisor
2338
2338 FC2338 move    usp,a5          ; a5 vom User-Stack nehmen
233A        move.l  (a5)+,-(a7)     ; und in System-Stack setzen
233C        move    a5,usp          ; USP anpassen
233E        movea.l (a7)+,a5        ; a5 wiederherstellen
2340        clr.l   -(a7)           ; Nächstes Stack-Langwort löschen
2342 FC2342 move.l  #F1E2D3C4,-(a7) ; Testlangwort auf Stack legen
2348        cmpi.l  #F1E2D3C4,(a7)+ ; und prüfen, ob vorhanden
234E        beq.s   FC235E          ; ok: ->
2350        movea.l #40000,a7       ; sonst System-Stack neu einrichten
2356        clr.l   -(a7)           ; Langwort 0 auf Stack
2358        clr.w   -(a7)           ; Wort 0 auf Stack
235A        clr.l   -(a7)           ; Langwort -1 auf Stack
235C        not.l   (a7)
235E FC235E movem.l d0-d7/a0-a6,-(a7) ; Register retten
2362        lea     3C(a7),a5       ; a5 := Stackadresse vor den Registern
2366        lea     -16(a7),a7      ; a7 um 22 erniedrigen
236A        movea.l a7,a4           ; a4 -> Anfang des Debug-Stackbereichs
236C        clr.l   12(a4)          ; Task-Zeiger löschen
2370        move.l  (a5)+,d3        ; d3 := Ausnahmevektornummer
2372        move.l  d3,E(a4)        ; in Stackbereich kopieren
2376        move.l  a5,A(a4)        ; a5 in Stackbereich kopieren
237A        move    usp,a0          ; User-Stackpointer
237C        move.l  a0,6(a4)        ; in Stackbereich kopieren
2380        bsr     FC0546          ; ---> Prozessortyp ermitteln
2384        tst.b   d0              ; M 68000?
2386        bne.s   FC2396          ; nein: ->
2388        cmpi.w  #3,d3           ; Vektornummer > 3?
```

```
238C        bgt.s    FC2396           ; ja: ->
238E        cmpi.w   #2,d3            ; Vektornummer < 2?
2392        blt.s    FC2396           ; ja: ->
2394        addq.l   #8,a5            ; bei Adreß- oder Busfehler
2396 FC2396 btst.b   #5,(a5)          ; war Supervisor-Mode gesetzt?
239A        bne.s    FC23A6           ; ja: ->
239C        movea.l 4,a0              ; a0 := SysBase
23A0        move.l   114(a0),12(a4)   ; ThisTask in Stackbereich kopieren
23A6 FC23A6 move.w   (a5)+,4(a4)      ; Statusregister in Stackbereich
23AA        move.l   (a5),0(a4)       ; Rückkehradresse in Stackbereich
23AE        movea.l #200,a6           ; a6 -> Debug-Arbeitsbereich
23B4        move.w   INTENAR,EA(a6)   ; Int Enable Bits in Arbeitsbereich
23BC        move.w   #801,d0          ; Serial Port Interrupt Bits
23C0        move.w   d0,INTENA        ; Interrupts sperren
23C6        bset.l   #F,d0            ; Bit 15 setzen
23CA        and.w    d0,EA(a6)        ; mit Int Enable Bits maskieren
23CE        bsr      FC21F8           ; ---> RawIOInit
23D2        lea      -E4(pc),a0       ; a0 := FC22F0: Text 'rom-wack'
23D6        bsr      FC2278           ; ---> PutString
23DA        move.l   a4,84(a6)        ; Zeiger auf Stackbereich in Arb.Bereich
23DE        moveq    #FE,d2           ; Bit 0 := 0, Bits 1 bis 31 := 1
23E0        and.l    0(a4),d2         ; Bit 0 in Rückkehradresse löschen
23E4        move.l   d2,0(a4)         ; Rückkehradresse in Stackbereich
23E8        move.l   d2,C(a6)         ; Rückkehradresse in Arbeitsbereich
23EC        cmpi.l   #2F,E(a4)        ; Auslösung durch Breakpoint?
23F4        bne.s    FC23FE           ; nein: ->
23F6        subq.l   #2,C(a6)         ; Aktuelle Adresse um 2 vermindern
23FA        bsr      FC28A2           ; ---> Breakpoint entfernen
23FE FC23FE move.l   C(a6),0(a4)      ; Aktuelle Adresse in Stack setzen
2404        movea.l a4,a1             ; a1 -> Debug-Stackbereich
2406        move.l   #FC2326,BC       ; Trap #15 auf FC2326 richten
240E        move.l   #FC231E,24       ; Step-Vektor auf FC231E richten
2416        move.l   #1000000,18(a6)  ; Ersatzwert für LIMIT speichern
241E        bsr      FC27EA           ; ---> Register ausgeben
2422        bsr      FC2B94           ; ---> Kommandoschleife
2426
2426 ;------ TAB-Taste oder CTRL-I: Programmschritt
2426
2426 FC2426 movea.l 84(a6),a0         ; a0 := Debug-Datenbasis
242A        move.w   4(a0),56(a0)     ; Statusregister in Systemstack
```

```
2430         ori.w    #8000,56(a0)         ; Step-Bit setzen
2436         bra.s    FC2452               ; --->
2438
2438 ;------ GO: Programm ab aktueller Adresse ausführen
2438
2438 FC2438  movea.l 84(a6),a0            ; a0 := Debug-Datenbasis
243C         move.l   C(a6),0(a0)          ; Aktuelle Adresse auf Debug-Stack
2442
2442 ;------ RESUME oder CTRL-D: Programmausführung fortsetzen
2442
2442 FC2442  movea.l 84(a6),a0            ; a0 := Debug-Datenbasis
2446         move.w   4(a0),56(a0)         ; Statusregister in Systemstack
244C         andi.w   #7FFF,56(a0)         ; Step-Bit rücksetzen
2452 FC2452  move.w   EA(a6),INTENA        ; Interrupts freigeben
245A         move.l   0(a0),58(a0)         ; Debug-PC in Systemstack
2460         movea.l 6(a0),a1             ; a1 := User-Stackpointer
2464         move     a1,usp               ; usp aus Debug-Stack belegen
2466         lea      16(a0),a7            ; ssp auf Registerliste setzen
246A         movem.l (a7)+,d0-d7/a0-a6    ; Register installieren
246E         addq.l   #4,a7                ; ssp auf Systemstack setzen
2470         rte
2472
2472 ;------ Debug-Arbeitsbereich initialisieren
2472
2472 FC2472  movea.l a6,a0                ; a0 := a6 -> Arbeitsbereich
2474         move.w   #75,d0               ; d0 := Wortzähler
2478 FC2478  clr.w    (a0)+                ; 118 Worte löschen
247A         dbra     d0,FC2478
247E         move.l   #FC3254,0(a6)        ; Anfang der Tastenkode-Tabelle
2486         move.l   #10,14(a6)           ; Rahmen-Größe
248E         move.w   #4E4F,88(a6)         ; Opcode für Breakpoint (Trap #15)
2494         rts
2496
2496         DC.W     0
2498
2498 ;------
2498
2498         movea.l 4(a7),a0             ; Diese Routinen werden nicht
249C         move.l   (a0),d0              ; gerufen
249E         rts
```

```
24A0
24A0        movea.l 4(a7),a0
24A4        move.w  (a0),d0
24A6        rts
24A8
24A8        movem.l 4(a7),a0-a1
24AE        move.w  a1,(a0)
24B0        rts
24B2
24B2 ;------ user: Rückkehr zu Multitasking
24B2
24B2 FC24B2 movea.l 84(a6),a0          ; a0 := Debug-Datenbasis
24B6        btst.b  #5,4(a0)           ; War Supervisor-Bit gesetzt?
24BC        bne.s   FC24EC             ; ja: ->
24BE        movea.l 6(a0),a1           ; a1 := User-Stackpointer
24C2        lea     -5C(a1),a1         ; um 92 vermindern
24C6        move.l  a1,84(a6)          ; und als Debug-Basis speichern
24CA        lea     5C(a1),a1          ; alten Wert wiederherstellen
24CE        adda.w  #5C,a0             ; a0 -> Systemstack über Rückkehradr.
24D2        move.l  a0,d1              ; d1 := a0
24D4        move.l  a0,d0              ; d0 := a0
24D6        sub.l   a7,d0              ; d0 := d0 - System-Stackpointer
24D8        bra.s   FC24DC             ; --->
24DA
24DA FC24DA move.b  -(a0),-(a1)        ; Debug-Stack in User-Stack kopieren
24DC FC24DC dbra    d0,FC24DA
24E0        movea.l d1,a7              ; SSP -> Systemstack über Rückkehradr.
24E2        move    a1,usp             ; USP wiederherstellen
24E4        movea.l 84(a6),a0          ; a0 := Debug-Datenbasis
24E8        move    4(a0),sr           ; Statusregister wiederherstellen
24EC FC24EC bsr     FC2A72             ; ---> Neue Zeile
24F0        rts
24F2
24F2 ;------ Text im Puffer mit Namen in Tabelle vergleichen
24F2
24F2        movem.l 4(a7),a0-a1        ; a0 -> Tabelle, a1 -> Kommandoname
24F8 FC24F8 moveq   #FF,d0             ; d0 als Zähler initialisieren
24FA FC24FA move.b  (a0)+,d1           ; d1 := Zeichen aus Tabelle
24FC        beq.s   FC2508             ; Ende des Namens: ->
24FE        cmp.b   (a1)+,d1           ; mit Zeichen in Puffer vergleichen
```

```
2500        dbne    d0,FC24FA         ; bis Ungleichheit auftritt: ->
2504        neg.l   d0                ; Zählerwert negieren
2506        bra.s   FC250E            ; --->
2508
2508 FC2508 cmp.b   (a1)+,d1          ; Kommandoname auch zu Ende?
250A        bne.s   FC250E            ; nein: zurück mit d0 <> 0
250C        moveq   #0,d0             ; Flag für Übereinstimmung
250E FC250E rts
2510
2510 ;------ Taste RETURN ohne Eingabe
2510
2510 FC2510 move.b  #1,1E(a6)         ; Ausgabeflag setzen
2516        rts
2518
2518 ;------
2518
2518        move.l  8(a6),d0          ; Routine wird nicht gerufen
251C        move.l  d0,-(a7)
251E        addq.l  #4,a7
2520        rts
2522
2522 ;------ Taste '>' oder SPACE: Ein Wort vor
2522
2522 FC2522 addq.l  #2,C(a6)          ; Aktuelle Adresse um 2 erhöhen
2526        move.b  #1,1E(a6)         ; Ausgabeflag setzen
252C        btst.b  #1,1F(a6)         ; Änderungsmodus aktiv?
2532        beq.s   FC2540            ; nein: ->
2534        bsr     FC2A72            ; ---> Neue Zeile
2538        bsr     FC261A            ; ---> Adresse und Wort ausgeben
253C        clr.b   1E(a6)            ; Ausgabeflag löschen
2540 FC2540 rts
2542
2542 ;------ Taste '<' oder BACKSPACE: Ein Wort zurück
2542
2542 FC2542 subq.l  #2,C(a6)          ; Aktuelle Adresse um 2 vermindern
2546        move.b  #1,1E(a6)         ; Ausgabeflag setzen
254C        btst.b  #1,1F(a6)         ; Änderungsmodus aktiv?
2552        beq.s   FC2560            ; nein: ->
2554        bsr     FC2A72            ; ---> Neue Zeile
2558        bsr     FC261A            ; ---> Adresse und Wort ausgeben
```

```
255C         clr.b    1E(a6)          ; Ausgabeflag löschen
2560 FC2560  rts
2562
2562 ;------ Taste '.': Einen Rahmen vor
2562
2562 FC2562  move.l   14(a6),d0       ; d0 := Rahmengröße
2566         add.l    d0,C(a6)        ; zur aktuellen Adresse addieren
256A         move.b   #1,1E(a6)       ; Ausgabeflag setzen
2570         rts
2572
2572 ;------ Taste ',': Einen Rahmen zurück
2572
2572 FC2572  move.l   14(a6),d0       ; d0 := Rahmengröße
2576         sub.l    d0,C(a6)        ; von aktueller Adresse subtrahieren
257A         move.b   #1,1E(a6)       ; Ausgabeflag setzen
2580         rts
2582
2582 ;------ Taste '[': Indirektion vor
2582
2582 FC2582  movea.l 26(a6),a1        ; a1 := Indirektion-Zeiger
2586         movea.l C(a6),a0         ; a0 := aktuelle Adresse
258A         move.l   a0,(a1)+        ; in Indirektion-Stack schreiben
258C         move.l   a1,26(a6)       ; Indirektion-Zeiger zurückschreiben
2590         moveq    #FE,d0          ; um ggf. Bit 0 zu löschen
2592         and.l    (a0),d0         ; d0 := Inhalt von (a0)
2594         move.l   d0,C(a6)        ; als aktuelle Adresse übernehmen
2598         move.b   #1,1E(a6)       ; Ausgabeflag setzen
259E         rts
25A0
25A0 ;------ Taste ']': Indirektion zurück
25A0
25A0 FC25A0  movea.l 26(a6),a1        ; a1 := Indirektion-Zeiger
25A4         move.l   -(a1),C(a6)     ; alte aktuelle Adresse wiederherst
25A8         move.b   #1,1E(a6)       ; Ausgabeflag setzen
25AE         move.l   a1,26(a6)       ; Indirektion-Zeiger zurückschreiben
25B2         rts
25B4
25B4 ;------ Taste '+': n Bytes vor
25B4
25B4 FC25B4  moveq    #'+',d0
```

```
25B6         bsr      FC222E           ; ---> RawPutChar
25BA         bsr      FC2BF0           ; ---> Zahl n übernehmen
25BE         tst.l    d0               ; Leere Eingabe?
25C0         beq      FC2A72           ; ja: Neue Zeile ->
25C4         move.l   8(a6),d0         ; d0 := n
25C8         add.l    d0,C(a6)         ; zur aktuellen Adresse addieren
25CC         move.b   #1,1E(a6)        ; Ausgabeflag setzen
25D2         rts
25D4
25D4 ;------ Taste '-': n Bytes zurück
25D4
25D4 FC25D4  moveq    #'-',d0
25D6         bsr      FC222E           ; ---> RawPutChar
25DA         bsr      FC2BF0           ; ---> Zahl n übernehmen
25DE         tst.l    d0               ; Leere Eingabe?
25E0         beq      FC2A72           ; ja: Neue Zeile ->
25E4         move.l   8(a6),d0         ; d0 := n
25E8         sub.l    d0,C(a6)         ; von aktueller Adresse subtrahieren
25EC         move.b   #1,1E(a6)        ; Ausgabeflag setzen
25F2         rts
25F4
25F4 ;------ Aktuelle Adresse setzen (nicht gerufen)
25F4
25F4         move.l   8(a6),C(a6)      ; Aktuelle Adresse := Eingabe
25FA         move.b   #1,1E(a6)        ; Ausgabeflag setzen
2600         rts
2602
2602 ;------ Taste ':': Rahmengröße festlegen
2602
2602 FC2602  moveq    #':',d0
2604         bsr      FC222E           ; ---> RawPutChar
2608         bsr      FC2BF0           ; ---> Rahmengrüße übernehmen
260C         move.l   8(a6),14(a6)     ; Rahmengröße := Eingabe
2612         move.b   #1,1E(a6)        ; Ausgabeflag setzen
2618         rts
261A
261A ;------ Adresse, Wort und '=' ausgeben
261A
261A FC261A  move.l   C(a6),d0         ; d0 := Aktuelle Adresse
261E         bsr      FC2780           ; ---> 6-stellig ausgeben
```

```
2622          tst.l    14(a6)           ; Rahmengröße
2626          beq.s    FC2638           ; = 0: ->
2628          movea.l  d0,a0            ; a0 := d0 = aktuelle Adresse
262A          move.w   (a0),d0          ; d0 := Wort an dieser Adresse
262C          bsr      FC2788           ; ---> 4-stellig ausgeben
2630          moveq    #'=',d0
2632          bsr      FC222E           ; ---> RawPutChar
2636          bra.s    FC2640           ; --->
2638
2638 FC2638   lea      2E(pc),a0        ; a0 := FC2668 (Text 'xxxx =')
263C          bsr      FC2278           ; ---> PutString
2640 FC2640   rts
2642
2642 ;------ Taste '=': Wort im Speicher ändern
2642
2642 FC2642   bsr.s    FC261A           ; ---> Adresse und Wort ausgeben
2644          bsr      FC2BF0           ; ---> Änderungswort übernehmen
2648          tst.l    d0               ; Leere Eingabe?
264A          beq.s    FC2658           ; ja: ->
264C          movea.l  C(a6),a0         ; a0 := aktuelle Adresse
2650          move.l   8(a6),d0         ; d0 := eingegebenes Wort
2654          move.w   d0,(a0)          ; Wort an Adresse a0 abspeichern
2656          moveq    #1,d0            ; d0 := 1
2658 FC2658   btst.b   #1,1F(a6)        ; Änderungsmodus aktiv?
265E          bne.s    FC2666           ; ja: ->
2660          move.b   #1,1E(a6)        ; Ausgabeflag setzen
2666 FC2666   rts
2668
2668 ;------
2668
2668 FC2668   DC.B     'xxxx =',0,0
2670
2670 ;------ alter: Änderungsmodus aktivieren
2670
2670 FC2670   bset.b   #1,1F(a6)        ; Änderungsflag setzen
2676 FC2676   bsr      FC2A72           ; ---> Neue Zeile
267A          bsr.s    FC2642           ; ---> Änderungsroutine
267C          tst.l    d0               ; Abbruchkommando?
267E          beq.s    FC2686           ; ja: ->
2680          addq.l   #2,C(a6)         ; Aktuelle Adresse um 2 erhöhen
```

```
2684         bra.s    FC2676            ; ---> Loop
2686
2686 FC2686  bclr.b   #1,1F(a6)         ; Änderungsflag löschen
268C         bsr      FC2A72            ; ---> Neue Zeile
2690         rts
2692
2692 ;------ Ausgang bei unbekanntem Kommando (nicht gerufen)
2692
2692         lea      8(pc),a0          ; a0 := FC269C (Text 'not yet impl...')
2696         bsr      FC2278            ; ---> PutString
269A         rts
269C
269C FC269C  DC.B     0A,'not yet implemented',0A,0
26B2
26B2 ;------ list: System List ausgeben
26B2
26B2 FC26B2  move.l   a2,-(a7)          ; a2 auf Stack retten
26B4         movea.l  C(a6),a2          ; a2 := aktuelle Adresse
26B8         tst.l    4(a2)             ; Nächstes Langwort = 0?
26BC         bne.s    FC26C0            ; nein: kein List Header ->
26BE         movea.l  (a2),a2           ; a2 -> 1. Node
26C0 FC26C0  move.l   (a2),d0           ; d0 := ln_Succ
26C2         beq.s    FC26F4            ; Ende der Liste: ->
26C4         move.l   A(a2),d0          ; d0 -> Namensstring
26C8         bne.s    FC26D0            ; Name vorhanden: ->
26CA         move.l   #FC271E,d0        ; d0 -> Endemarke 0
26D0 FC26D0  move.l   d0,-(a7)          ; d0 auf Stack
26D2         moveq    #0,d0             ; d0 := 0
26D4         move.b   9(a2),d0          ; d0 := Priorität
26D8         move.l   d0,-(a7)          ; d0 auf Stack
26DA         move.b   8(a2),d0          ; d0 := Typ
26DE         move.l   d0,-(a7)          ; d0 auf Stack
26E0         pea      (a2)              ; Node-Adresse auf Stack
26E2         movea.l  a7,a1             ; a1 -> Ausgabedaten im Stack
26E4         lea      16(pc),a0         ; a0 := FC26FC (Formatstring)
26E8         bsr      FC22D2            ; ---> Daten formatiert ausgeben
26EC         lea      14(a7),a7         ; Stack wieder ausräumen
26F0         movea.l  (a2),a2           ; a2 -> nächster Node
26F2         bra.s    FC26C0            ; ---> Loop
26F4
```

```
26F4 FC26F4  bsr      FC2A72              ; ---> Neue Zeile
26F8         movea.l  (a7)+,a2            ; a2 wiederherstellen
26FA         rts
26FC
26FC FC26FC  DC.B     0A,'%061x  type %-21d  pri %-41d "%s"'
271E
271E FC271E  DC.W     0
2720
2720 ;------ Einen Rahmen ausgeben
2720
2720 FC2720  movem.l  d0/d2-d3/a2-a3,-(a7) ; Register retten
2724         link     a5,#-28             ; 40 Bytes Puffer im Stack einrichten
2728         bsr      FC2A72              ; ---> Neue Zeile
272C         movea.l  d0,a2               ; a2 := d0 = aktuelle Adresse
272E         move.l   d1,d2               ; d2 := d1 = Rahmengröße
2730         beq.s    FC2770              ; Rahmengröße = 0: fertig ->
2732 FC2732  move.l   a2,d0               ; d0 := a2 = aktuelle Adresse
2734         bsr.s    FC2780              ; ---> 6-stellig ausgeben
2736         moveq    #7,d3               ; Zähler für 8 Worte setzen
2738         lea      -20(a5),a3          ; a3 -> Zeichenpuffer
273C FC273C  move.w   (a2)+,d0            ; d0 := nächstes Wort
273E         bsr.s    FC2788              ; ---> 4-stellig ausgeben
2740         move.l   d0,-(a7)            ; d0 auf Stack retten
2742         lsr.w    #8,d0               ; oberes Byte in Byte-Position
2744         bsr      FC2ABC              ; ---> Zeichen generieren
2748         move.w   d0,(a3)+            ; und in Puffer setzen
274A         move.l   (a7)+,d0            ; d0 wiederherstellen
274C         bsr      FC2ABC              ; ---> Zeichen generieren
2750         move.w   d0,(a3)+            ; und in Puffer setzen
2752         subq.l   #2,d2               ; Rahmengröße um Wortlänge vermindern
2754         ble.s    FC2766              ; Rahmen fertig: ->
2756         dbra     d3,FC273C           ; nächstes Wort, maximal 8 pro Zeile ->
275A         clr.w    (a3)+               ; Puffer mit Kode 0 abschließen
275C         lea      -20(a5),a0          ; a0 -> Zeichenpuffer
2760         bsr      FC2A86              ; ---> PutString, Neue Zeile
2764         bra.s    FC2732              ; ---> Loop
2766
2766 FC2766  clr.w    (a3)+               ; Zeichenpuffer abschließen
2768         lea      -20(a5),a0          ; a0 -> Zeichenpuffer
276C         bsr      FC2A86              ; ---> PutString, Neue Zeile
```

```
2770 FC2770  unlk    a5                   ; Zeichenpuffer auflösen
2772         movem.l (a7)+,d0/d2-d3/a2-a3 ; Register wiederherstellen
2776         rts
2778
2778 ;------ Zahlenausgabe in Hex
2778
2778 FC2778  movem.l d0-d1/a0-a1,-(a7)   ; Register retten
277C         moveq   #8,d1                ; d1 := Stellenzahl
277E         bra.s   FC278E               ; --->
2780
2780 FC2780  movem.l d0-d1/a0-a1,-(a7)   ; Register retten
2784         moveq   #6,d1                ; d1 := Stellenzahl
2786         bra.s   FC278E               ; --->
2788
2788 FC2788  movem.l d0-d1/a0-a1,-(a7)   ; Register retten
278C         moveq   #4,d1                ; d1 := Stellenzahl
278E FC278E  bsr     FC2294               ; ---> PutHex
2792         movem.l (a7)+,d0-d1/a0-a1   ; Register wiederherstellen
2796         rts
2798
2798 ;------ Datenregister, Adreßregister, Stackrahmen ausgeben
2798
2798 FC2798  movem.l d2/a2,-(a7)         ; Register retten
279C         movea.l a0,a2                ; a2 := a0 -> Register im Debug-Stack
279E         moveq   #7,d2                ; Zähler für 8 Register
27A0         lea     C2(pc),a0            ; a0 := FC2864  (Text 'DR:')
27A4         bsr.s   FC27D8               ; ---> Text und Datenregister ausgeben
27A6         moveq   #6,d2                ; Zähler für 7 Register
27A8         lea     C0(pc),a0            ; a0 := FC286A  (Text 'AR:')
27AC         bsr.s   FC27D8               ; ---> Text und 7 Adreßregister ausgeben
27AE         addq.l  #4,a2                ; a2 auf System-Stack vorrücken
27B0         btst.b  #5,(a2)              ; War Supervisor-Bit gesetzt?
27B4         bne.s   FC27BC               ; ja: ->
27B6         suba.w  #50,a2               ; a2 -> USP im Debug-Stackbereich
27BA         movea.l (a2),a2              ; a2 := usp
27BC FC27BC  lea     B2(pc),a0            ; a0 := FC2870  (Text 'SF:')
27C0         bsr     FC2278               ; ---> PutString
27C4         moveq   #E,d2                ; Zähler für 15 Worte im Stack
27C6 FC27C6  move.w  (a2)+,d0             ; d0 := nächstes Wort im Stack
27C8         bsr.s   FC2788               ; ---> 4-stellig ausgeben
```

```
27CA        dbra     d2,FC27C6         ; bis 15 Worte ausgegeben: ->
27CE        bsr      FC2A72            ; ---> Neue Zeile
27D2        movem.l  (a7)+,d2/a2       ; Register wiederherstellen
27D6        rts
27D8
27D8 FC27D8 bsr      FC2278            ; ---> PutString
27DC FC27DC move.l   (a2)+,dØ          ; dØ := Registerinhalt
'27DE       bsr.s    FC2778            ; ---> 8-stellig ausgeben
27EØ        dbra     d2,FC27DC         ; bis alle Register ausgegeben: ->
27E4        rts
27E6
27E6 ;------ regs: Register-Ausgabe
27E6
27E6 FC27E6 movea.l  84(a6),a1         ; a1 -> Debug-Datenbasis
27EA FC27EA move.l   a1,-(a7)          ; a1 auf Stack retten
27EC        lea      E(pc),aØ          ; aØ := FC27FC  (Formatstring)
27FØ        bsr      FC22D2            ; ---> Daten formatiert ausgeben
27F4        movea.l  (a7)+,a1          ; a1 wiederherstellen
27F6        lea      16(a1),aØ         ; aØ -> Registerliste im Stack
27FA        bra.s    FC2798            ; ---> zur Ausgabe
27FC
27FC ;------ Formatstrings für Rahmenausgabe
27FC
27FC FC27FC DC.B     ØA,'PC: %Ø61x  SR: %Ø4x  USP: %Ø61x'
281C        DC.B     '  SSP: %Ø61x  XCPT: %Ø41x  TASK: %Ø61x',Ø
281C
2843        DC.B     ØA,'PC: %Ø61x  SR: %Ø4x  USP: %Ø61x',Ø'
2843
2864 FC2864 DC.B     ØA,'DR: ',Ø
286A
286A FC286A DC.B     ØA,'AR: ',Ø
287Ø
287Ø FC287Ø DC.B     ØA,'SF: ',Ø
2876
2876 ;------ d2.w ab Adresse dØ d1+1 mal abspeichern (nicht gerufen)
2876
2876        move.l   a2,-(a7)          ; a2 retten
2878        movea.l  dØ,a2             ; a2 := dØ = Anfangsadresse
287A        bra.s    FC287E            ; --->
287C
```

```
287C FC287C  move.w  d2,(a2)+          ; d2 speichern
287E FC287E  dbra    d1,FC287C         ; d1 mal wiederholen
2882         movea.l (a7)+,a2          ; a2 wiederherstellen
2884         rts
2886
2886 ;------ Breakpoint in Tabelle suchen
2886
2886 FC2886  bsr     FC2A72            ; ---> Neue Zeile
288A         lea     8A(a6),aØ         ; aØ -> Breakpoint-Tabelle
288E         moveq   #F,d1             ; d1 := Zähler für 16 Breakpoints
289Ø FC289Ø  cmpa.l  (aØ),a1           ; a1 = Breakpoint-Adresse?
2892         beq.s   FC289E            ; ja: ->
2894         addq.l  #6,aØ             ; aØ auf nächsten Tabelleneintrag setzen
2896         dbra    d1,FC289Ø         ; und weitersuchen bis Tabellenende ->
289A         moveq   #Ø,dØ             ; dØ := Ø: Flag 'nicht gefunden'
289C         rts
289E
289E FC289E  move.l  aØ,dØ             ; dØ := aØ -> Breakpoint in Tabelle
28AØ         rts
28A2
28A2 ;------ clear: Breakpoint an aktueller Adresse entfernen
28A2
28A2 FC28A2  movea.l C(a6),a1          ; a1 := aktuelle Adresse
28A6         bsr.s   FC2886            ; ---> Breakpoint in Tabelle suchen
28A8         beq.s   FC28BØ            ; nicht gefunden: ->
28AA         clr.l   (aØ)              ; Adresse in Tabelle löschen
28AC         move.w  4(aØ),(a1)        ; alten Opcode wiederherstellen
28BØ FC28BØ  rts
28B2
28B2 ;------ reset: Alle Breakpoints entfernen
28B2
28B2 FC28B2  lea     8A(a6),a1         ; a1 -> Breakpoint-Tabelle
28B6         moveq   #F,d1             ; d1 := Zähler für 16 Breakpoints
28B8 FC28B8  move.l  (a1),dØ           ; dØ := Breakpoint-Adresse aus Tabelle
28BA         beq.s   FC28C4            ; Kein Eintrag: ->
28BC         movea.l dØ,aØ             ; aØ := dØ = Breakpoint-Adresse
28BE         clr.l   (a1)              ; Adresse in Tabelle löschen
28CØ         move.w  4(a1),(aØ)        ; alten Opcode wiederherstellen
28C4 FC28C4  addq.l  #6,a1             ; Tabellenzeiger vorrücken
28C6         dbra    d1,FC28B8         ; wiederholen bis Tabellenende ->
```

```
28CA        bsr     FC2A72          ; ---> Neue Zeile
28CE        rts
28DØ
28DØ ;------ set: Breakpoint an aktueller Adresse setzen
28DØ
28DØ FC28DØ movea.l C(a6),a1        ; a1 := aktuelle Adresse
28D4        bsr.s   FC2886          ; ---> in Breakpoint-Tabelle suchen
28D6        bne.s   FC28FC          ; Breakpoint bereits gesetzt: ->
28D8        lea     8A(a6),aØ       ; aØ -> Breakpoint-Tabelle
28DC        moveq   #F,d1           ; d1 := Zähler für 16 Einträge
28DE FC28DE tst.l   (aØ)            ; Tabelleneintrag belegt?
28EØ        beq.s   FC28F2          ; nein: ->
28E2        addq.l  #6,aØ           ; Zeiger auf nächsten Eintrag setzen
28E4        dbra    d1,FC28DE       ; wiederholen bis Tabellenende ->
28E8        lea     14(pc),aØ       ; aØ := FC28FE  (Text 'too many')
28EC        bsr     FC2278          ; ---> PutString
28FØ        bra.s   FC28FC          ; --->
28F2
28F2 FC28F2 move.w  (a1),4(aØ)      ; Opcode in Tabelle schreiben
28F6        move.w  88(a6),(a1)     ; und durch 'TRAP #15' ersetzen
28FA        move.l  a1,(aØ)         ; Adresse in Tabelle schreiben
28FC FC28FC rts
28FE
28FE FC28FE DC.B    ØA,'too many',ØA,Ø,Ø
29ØA
29ØA ;------ show: Alle Breakpoint-Adressen ausgeben
29ØA
29ØA FC29ØA lea     8A(a6),aØ       ; aØ -> Breakpoint-Tabelle
29ØE        moveq   #F,d1           ; d1 := Zähler für 16 Einträge
291Ø FC291Ø move.l  (aØ),dØ         ; dØ := Breakpoint-Adresse
2912        beq.s   FC291C          ; nicht belegt: ->
2914        bsr     FC2A72          ; ---> Neue Zeile
2918        bsr     FC278Ø          ; ---> Adresse 6-stellig ausgeben
291C FC291C addq.l  #6,aØ           ; Zeiger auf nächsten Eintrag setzen
291E        dbra    d1,FC291Ø       ; wiederholen bis Tabellenende ->
2922        bsr     FC2A72          ; ---> Neue Zeile
2926        rts
2928
2928        rts
292A
```

```
292A ;------ Warten auf Tasteneingabe mit Echo
292A
292A FC292A  bsr      FC2222              ; ---> Warten auf Tasteneingabe
292E         move.l   d0,-(a7)            ; Zeichen auf Stack retten
2930         bsr      FC222E              ; ---> RawPutChar
2934         move.l   (a7)+,d0            ; Zeichen wiederherstellen
2936         rts
2938
2938 ;------ Taste '!': Register ändern
2938
2938 FC2938  movem.l  d2/a2,-(a7)         ; Register retten
293C         moveq    #'!',d0
293E         bsr      FC222E              ; ---> RawPutChar
2942         bsr.s    FC292A              ; ---> Warten auf Eingabe mit Echo
2944         bsr      FC2D48              ; ---> Klein-Großbuchstaben-Wandlung
2948         movea.l  84(a6),a1           ; a1 := Debug-Datenbasis
294C         lea      16(a1),a0           ; a0 -> Datenregisterliste
2950         moveq    #7,d2               ; d2 := Zähler für 8 Datenregister
2952         cmpi.b   #'D',d0             ; Datenregister zu ändern?
2956         beq.s    FC2970              ; ja: ->
2958         lea      20(a0),a0           ; a0 -> Adreßregisterliste
295C         moveq    #6,d2               ; d2 := Zähler für 7 Adreßregister
295E         cmpi.b   #'A',d0             ; Adreßregister zu ändern?
2962         beq.s    FC2970              ; ja: ->
2964         lea      6(a1),a2            ; a2 -> USP im Debug-Stackbereich
2968         cmpi.b   #'U',d0             ; User-Stackpointer zu ändern?
296C         beq.s    FC298C              ; ja: ->
296E         bra.s    FC29A8              ; --->
2970
2970 FC2970  bsr.s    FC292A              ; ---> Warten auf Eingabe mit Echo
2972         cmpi.w   #8,d0               ; BACKSPACE eingegeben?
2976         beq.s    FC29A8              ; ja: ->
2978         bsr      FC2AAC              ; ---> Prüfen ob Ziffer
297C         bne.s    FC29A8              ; nein: ->
297E         subi.w   #30,d0              ; d0 := Wert der eingegebenen Ziffer
2982         cmp.b    d0,d2               ; Registernummer im zulässigen Bereich?
2984         blt.s    FC29A8              ; nein: ->
2986         lsl.w    #2,d0               ; mal 4 ergibt Offset in Registerliste
2988         lea      0(a0,d0.w),a2       ; a2 -> Register in der Liste
298C FC298C  move.l   (a2),d0             ; d0 := Inhalt des Registers
```

```
298E         bsr      FC2A62              ; ---> PutSpace
2992         bsr      FC2778              ; ---> Registerinhalt 8-stellig ausgeben
2996         moveq    #'=',d0
2998         bsr      FC222E              ; ---> RawPutChar
299C         bsr      FC2BF0              ; ---> Zahleneingabe übernehmen
29A0         tst.b    d0                  ; Leere Eingabe?
29A2         beq.s    FC29A8              ; ja: ->
29A4         move.l   8(a6),(a2)          ; Eingabe in Registerliste eintragen
29A8 FC29A8  bsr      FC2A72              ; ---> Neue Zeile
29AC         movem.l  (a7)+,d2/a2         ; Register wiederherstellen
29B0         rts
29B2
29B2 ;------ Taste '^' oder LIMIT: Grenze für FIND und FILL festlegen
29B2
29B2 FC29B2  move.l   C(a6),18(a6)        ; Eingabe als Limit eintragen
29B8         move.b   #1,1E(a6)           ; Ausgabeflag setzen
29BE         rts
29C0
29C0 ;------ find: Kodemuster suchen
29C0
29C0 FC29C0  movem.l  a2-a3,-(a7)         ; Register retten
29C4         bsr.s    FC2A04              ; ---> Muster übernehmen
29C6         beq.s    FC29D4              ; Leere Eingabe: ->
29C8         bsr.s    FC29E0              ; ---> Muster suchen
29CA         beq.s    FC29D4              ;
29CC         bclr.l   #0,d0
29D0         move.l   d0,C(a6)
29D4 FC29D4  move.b   #1,1E(a6)           ; Ausgabeflag setzen
29DA         movem.l  (a7)+,a2-a3         ; Register wiederherstellen
29DE         rts
29E0
29E0 FC29E0  move.l   a4,-(a7)            ; a4 auf Stack retten
29E2         lea      -2(a0),a4           ; a4 := aktuelle Adresse-2
29E6 FC29E6  move.l   d0,d1               ; d1 := d0 = Anzahl der Bytes im Muster
29E8         movea.l  a3,a1               ; a1 := a3 -> Anfang des Musters
29EA         addq.l   #1,a4               ; Suchzeiger inkrementieren
29EC         movea.l  a4,a0               ; a0 := a4 = Suchzeiger
29EE FC29EE  cmpa.l   a0,a2               ; Suchzeiger < Limit?
29F0         ble.s    FC29FE              ; nein: ->
29F2         cmpm.b   (a1)+,(a0)+         ; Byte aus Muster = Byte im Speicher?
```

```
29F4          bne.s   FC29E6              ; nein: ->
29F6          subq.w  #1,d1               ; Bytezähler dekrementieren
29F8          bgt.s   FC29EE              ; noch nicht Null: ->
29FA          move.l  a4,d0               ; d0 := aktueller Suchzeiger
29FC          bra.s   FC2A00              ; --->
29FE
29FE FC29FE   moveq   #0,d0               ; d0 := 0 heißt: nicht gefunden
2A00 FC2A00   movea.l (a7)+,a4            ; a4 wiederherstellen
2A02          rts
2A04
2A04 FC2A04   lea     26(pc),a0           ; a0 := FC2A2A  (Text 'pattern?')
2A08          bsr     FC2278              ; ---> PutString
2A0C          bsr     FC2BF0              ; ---> Kode-Eingabe
2A10          tst.w   d0                  ; Leere Eingabe?
2A12          beq.s   FC2A2A              ; ja: ->
2A14          addq.w  #1,d0               ; Stellenzahl um 1 vergrößern
2A16          lsr.w   #1,d0               ; und halbieren ergibt Zahl der Bytes
2A18          moveq   #4,d1               ; Zahl steht im Speicher rechtsbündig,
2A1A          sub.w   d0,d1               ; daher Bytezahl von 4 subtrahieren
2A1C          lea     8(a6,d1.w),a3       ; a3 -> Anfang des Kodemusters
2A20          movea.l C(a6),a0            ; a0 := aktuelle Adresse
2A24          movea.l 18(a6),a2           ; a2 := Limit
2A28          moveq   #1,d1               ; bewirkt ZF := 0
2A2A FC2A2A   rts
2A2C
2A2C          DC.B    ' pattern? ',0,0
2A38
2A38 ;------ fill: Speicherbereich mit Kodemuster füllen
2A38
2A38 FC2A38   movem.l a2-a3,-(a7)         ; Register retten
2A3C          bsr.s   FC2A04              ; ---> Muster übernehmen
2A3E          beq.s   FC2A42              ; Leere Eingabe: ->
2A40          bsr.s   FC2A4E              ; ---> Muster in Speicher schreiben
2A42 FC2A42   move.b  #1,1E(a6)           ; Ausgabeflag setzen
2A48          movem.l (a7)+,a2-a3         ; Register wiederherstellen
2A4C          rts
2A4E
2A4E FC2A4E   subq.l  #1,d0               ; Stellenzahl um 1 vermindern
2A50 FC2A50   move.l  d0,d1               ; d1 := Zähler
2A52          movea.l a3,a1               ; a1 := a3 -> Anfang des Musters
```

```
2A54 FC2A54  cmpa.l  aØ,a2              ; Aktuelle Adresse < Limit?
2A56         ble.s   FC2A6Ø             ; nein: fertig ->
2A58         move.b  (a1)+,(aØ)+        ; Byte aus Muster übertragen
2A5A         dbra    d1,FC2A54          ; wiederholen bis Muster fertig ->
2A5E         bra.s   FC2A5Ø             ; ---> Loop
2A6Ø
2A6Ø FC2A6Ø  rts
2A62
2A62 ;------ SPACE ausgeben
2A62
2A62 FC2A62  movem.l dØ-d1/aØ-a1,-(a7)  ; Register retten
2A66         moveq   #' ',dØ
2A68         bsr     FC222E             ; ---> RawPutChar
2A6C         movem.l (a7)+,dØ-d1/aØ-a1  ; Register wiederherstellen
2A7Ø         rts
2A72
2A72 ;------ Neue Zeile
2A72
2A72 FC2A72  movem.l dØ-d1/aØ-a1,-(a7)  ; Register retten
2A76         lea     C(pc),aØ           ; aØ := FC2A84  (Line Feed Kode)
2A7A         bsr     FC2278             ; ---> RawPutChar
2A7E         movem.l (a7)+,dØ-d1/aØ-a1  ; Register wiederherstellen
2A82         rts
2A84
2A84 FC2A84  DC.B    ØA,Ø               ; LineFeed Kode
2A86
2A86 ;------ String ausgeben, neue Zeile
2A86
2A86 FC2A86  movem.l dØ-d1/aØ-a1,-(a7)  ; Register retten
2A8A         bsr     FC2278             ; ---> PutString
2A8E         bsr.s   FC2A72             ; ---> Neue Zeile
2A9Ø         movem.l (a7)+,dØ-d1/aØ-a1  ; Register wiederherstellen
2A94         rts
2A96
2A96 ;------
2A96
2A96         moveq   #D,dØ
2A98         rts
2A9A
2A9A ;------ Zeichenempfang (nicht gerufen)
```

```
2A9A
2A9A        movem.l d0-d1/a0-a1,-(a7)  ; Register retten
2A9E        bsr     FC2202             ; ---> RawMayGetChar
2AA2        cmpi.w  #-1,d0             ; Kein Zeichen empfangen?
2AA6        movem.l (a7)+,d0-d1/a0-a1  ; Register wiederherstellen
2AAA        rts
2AAC
2AAC ;------ ChkNum: Prüfen, ob Ziffer in d0
2AAC
2AAC FC2AAC cmpi.b  #'0',d0            ; d0 < '0'?
2AB0        blt.s   FC2ABA             ; ja: ->
2AB2        cmpi.b  #'9',d0            ; d0 > '9'?
2AB6        bgt.s   FC2ABA             ; ja: ->
2AB8        cmp.b   d0,d0              ; ergibt ZF = 1
2ABA FC2ABA rts
2ABC
2ABC ;------ Zeichen aus Kode erzeugen für Rahmenausgabe
2ABC
2ABC FC2ABC move.l  d2,-(a7)           ; d2 auf Stack retten
2ABE        move.w  #'..',d2           ; d2 für nicht druckbare Kodes belegen
2AC2        tst.b   d0                 ; d0 = Zeichenkode
2AC4        beq.s   FC2AE2             ; = 0: nicht druckbar ->
2AC6        btst.l  #7,d0              ; Bit 7 gesetzt?
2ACA        bne.s   FC2AE2             ; ja: nicht druckbar ->
2ACC        move.w  #2000,d2           ; Erstes Zeichen := Leerstelle
2AD0        move.b  d0,d1              ; d1 := d0 = Zeichenkode
2AD2        andi.b  #E0,d1             ; Kode < $20?
2AD6        bne.s   FC2AE0             ; nein: druckbar ->
2AD8        move.w  #5E00,d2           ; Erstes Zeichen '^' für CTRL
2ADC        ori.b   #40,d0             ; Buchstabenkode erzeugen
2AE0 FC2AE0 move.b  d0,d2              ; Zweites Zeichen einsetzen
2AE2 FC2AE2 move.l  d2,d0              ; d0 := Zeichen für Ausgabe
2AE4        move.l  (a7)+,d2           ; d2 wiederherstellen
2AE6        rts
2AE8
2AE8 ;------ Kommando-Routine ausführen
2AE8
2AE8 FC2AE8 clr.b   1E(a6)             ; Ausgabeflag löschen
2AEC        movea.l 4(a7),a0           ; a0 -> Kommandoroutine
2AF0        jsr     (a0)               ; ---> Routine ausführen
```

```
2AF2        tst.b   1E(a6)          ; Ausgabeflag gesetzt?
2AF6        beq.s   FC2B04          ; nein: fertig ->
2AF8        move.l  C(a6),d0        ; d0 := aktuelle Adresse
2AFC        move.l  14(a6),d1       ; d1 := Rahmengröße
2B00        bsr     FC2720          ; ---> Rahmen ausgeben
2B04 FC2B04 rts
2B06
2B06 ;------ Taste '?': Alle Kommandos ausgeben
2B06
2B06 FC2B06 move.l  a2,-(a7)        ; a2 retten
2B08        movea.l #FC33A6,a2      ; a2 -> Kommando-Tabelle
2B0E FC2B0E move.b  (a2)+,d0        ; Zeichen aus Tabelle
2B10        beq.s   FC2B18          ; Ende des Kommandonamens: ->
2B12        bsr     FC222E          ; ---> RawPutChar
2B16        bra.s   FC2B0E          ; ---> Loop
2B18
2B18 FC2B18 bsr     FC2A62          ; ---> PutSpace
2B1C        tst.b   (a2)            ; Folgt weiterer Name?
2B1E        bne.s   FC2B0E          ; ja: ->
2B20        bsr     FC2A72          ; ---> Neue Zeile
2B24        movea.l (a7)+,a2        ; a2 wiederherstellen
2B26        rts
2B28
2B28 ;------ Tastenkode in Tabelle suchen
2B28
2B28 FC2B28 move.l  d0,d1           ; d1 := d0 = Tastenkode
2B2A        movea.l 0(a6),a0        ; a0 -> Tabelle
2B2E        bra.s   FC2B3C          ; --->
2B30
2B30 FC2B30 move.l  (a0),d0         ; d0 := nächster Tabelleneintrag
2B32        andi.l  #FFFFFF,d0      ; Bits 24 bis 31 löschen
2B38        beq.s   FC2B4C          ; Tabellenende: ->
2B3A        movea.l d0,a0           ; a0 -> nächster Tabelleneintrag
2B3C FC2B3C cmp.b   4(a0),d1        ; Tastenkode gefunden?
2B40        beq.s   FC2B4A          ; ja: ->
2B42        blt.s   FC2B30          ; Liegt er zwischen dem ersten
2B44        cmp.b   5(a0),d1        ; und dem zweiten Tabellenkode?
2B48        bgt.s   FC2B30          ; nein: weitersuchen ->
2B4A FC2B4A move.l  a0,d0           ; d0 -> Tabelleneintrag
2B4C FC2B4C rts
```

```
2B4E
2B4E ;------ Tastenkode in Tabelle suchen, zugehörige Routine aufrufen
2B4E
2B4E         move.l  4(a7),d0           ; d0 := Tastenkode
2B52 FC2B52  bsr.s   FC2B28             ; ---> Kode in Tabelle suchen
2B54         tst.l   d0                 ; gefunden?
2B56         beq.s   FC2B64             ; nein: ->
2B58         movea.l d0,a0              ; a0 -> Tabelleneintrag
2B5A         move.l  6(a0),-(a7)        ; Routinenadresse auf Stack
2B5E         bsr     FC2AE8             ; ---> Routine ausführen
2B62         addq.l  #4,a7              ; Adresse vom Stack nehmen
2B64 FC2B64  rts
2B66
2B66 ;------ Eingabe in Kommandotabelle suchen
2B66
2B66         movea.l 4(a7),a0           ; a0 -> Eingabepuffer
2B6A FC2B6A  movem.l a2-a3,-(a7)        ; Register retten
2B6E         movea.l a0,a2              ; a2 := a0 -> Eingabepuffer
2B70         lea     FC33F4,a3          ; a3 -> Kommandotabelle
2B76         bra.s   FC2B7E             ; --->
2B78
2B78 FC2B78  move.l  (a3),d0            ; d0 -> nächster Tabelleneintrag
2B7A         beq.s   FC2B8E             ; Tabellenende: ->
2B7C         movea.l d0,a3              ; a3 := d0 -> nächster Tabelleneintrag
2B7E FC2B7E  movea.l 4(a3),a0           ; a0 -> Kommandoname in Namenstabelle
2B82         movea.l a2,a1              ; a1 := a2 -> Eingabepuffer
2B84         bsr     FC24F8             ; ---> Namen mit Eingabe vergleichen
2B88         tst.l   d0                 ; Übereinstimmung?
2B8A         bne.s   FC2B78             ; nein: weitersuchen ->
2B8C         move.l  a3,d0              ; d0 := a3 -> Tabelleneintrag
2B8E FC2B8E  movem.l (a7)+,a2-a3        ; Register wiederherstellen
2B92         rts
2B94
2B94 ;------ K o m m a n d o s c h l e i f e
2B94
2B94 FC2B94  bsr     FC2222             ; ---> Auf Tasteneingabe warten
2B98         move.b  d0,82(a6)          ; Tastenkode abspeichern
2B9C         bsr.s   FC2B52             ; ---> Tastenkode bearbeiten
2B9E         bra.s   FC2B94             ; ---> Loop
2BA0
```

```
2BA0 ;------ Tasten '_', '0'...'9', 'a'...'z', 'A'...'Z'
2BA0
2BA0 FC2BA0  lea     FC3334,a0            ; a0 -> Editier- und Texteingabetabelle
2BA6         cmpa.l  0(a6),a0             ; ist dies die aktuelle Tabelle?
2BAA         beq.s   FC2BBE               ; ja: ->
2BAC         clr.w   1C(a6)               ; Pufferzeiger := 0
2BB0         move.l  0(a6),4(a6)          ; Zeiger auf Kdo-Tastentabelle auslagern
2BB6         move.l  #FC3334,0(a6)        ; Tabelle aktuell machen
2BBE FC2BBE  cmpi.b  #' ',82(a6)          ; SPACE eingegeben?
2BC4         beq.s   FC2BEA               ; ja: ->
2BC6         move.w  1C(a6),d0            ; d0 := Pufferzeiger
2BCA         cmpi.w  #32,d0               ; Pufferende erreicht?
2BCE         bge.s   FC2BEA               ; ja: ->
2BD0         move.b  82(a6),d0            ; d0 := letzter Tastenkode
2BD4         bsr     FC222E               ; ---> RawPutChar
2BD8         lea     50(a6),a0            ; a0 -> Eingabepuffer
2BDC         move.w  1C(a6),d0            ; d0 := Pufferzeiger
2BE0         move.b  82(a6),0(a0,d0.w)    ; Tastenkode in Puffer schreiben
2BE6         addq.w  #1,1C(a6)            ; Pufferzeiger inkrementieren
2BEA FC2BEA  clr.w   20(a6)               ; Flag 'Puffer leer' löschen
2BEE         rts
2BF0
2BF0 ;------ Parametereingabe nach Kommando
2BF0
2BF0 FC2BF0  move.b  #' ',82(a6)          ; Leertaste simulieren
2BF6         bsr.s   FC2BA0               ; ---> Tabellenzeiger einrichten
2BF8         move.w  #1,24(a6)            ; Flag 'Parametereingabe' setzen
2BFE FC2BFE  cmpi.l  #FC3334,0(a6)        ; RETURN gedrückt?
2C06         bne.s   FC2C16               ; ja: ->
2C08         bsr     FC2222               ; ---> Auf Tasteneingabe warten
2C0C         move.b  d0,82(a6)            ; Tastenkode abspeichern
2C10         bsr     FC2B52               ; ---> Routine ausführen
2C14         bra.s   FC2BFE               ; ---> Loop
2C16
2C16 FC2C16  clr.w   24(a6)               ; Flag 'Parametereingabe' löschen
2C1A         moveq   #0,d0                ; d0 := 0
2C1C         move.b  22(a6),d0            ; d0 := Stellenzahl der Eingabe
2C20         tst.w   20(a6)               ; Flag 'Puffer leer' gesetzt?
2C24         beq.s   FC2C28               ; nein: ->
2C26         moveq   #0,d0                ; d0 := 0 für leere Eingabe
```

```
2C28 FC2C28  rts
2C2A
2C2A ;------ Taste SPACE
2C2A
2C2A FC2C2A  rts
2C2C
2C2C ;------ Tasten CTRL-X, CTRL-U: Ganzen Eingabepuffer löschen
2C2C
2C2C FC2C2C  move.w  #FFFF,1C(a6)        ; Pufferzeiger := -1
2C32         move.l  4(a6),Ø(a6)         ; Tabellenzeiger rücksetzen
2C38         bsr     FC2A72              ; ---> Neue Zeile
2C3C         move.w  #1,2Ø(a6)           ; Flag 'Puffer leer' setzen
2C42         rts
2C44
2C44 ;------ Taste BACKSPACE
2C44
2C44 FC2C44  tst.w   1C(a6)              ; Pufferzeiger prüfen
2C48         ble.s   FC2C2C              ; nicht positiv: ->
2C4A         lea     5Ø(a6),aØ           ; aØ -> Eingabepuffer
2C4E         move.w  1C(a6),dØ           ; dØ := Pufferzeiger
2C52         clr.b   Ø(aØ,dØ.w)          ; Endekode schreiben
2C56         subq.w  #1,1C(a6)           ; Pufferzeiger dekrementieren
2C5A         lea     E(pc),aØ            ; aØ := FC2C6A
2C5E         bsr     FC2278              ; ---> PutString
2C62         tst.w   1C(a6)              ; Pufferzeiger prüfen
2C66         ble.s   FC2C2C              ; nicht positiv: ->
2C68         rts
2C6A
2C6A FC2C6A  DC.B    8,2Ø,8,Ø
2C6E
2C6E ;------ Taste RETURN nach Eingabe
2C6E
2C6E FC2C6E  move.l  4(a6),Ø(a6)         ; Ø(a6) -> 1. Tastenkodetabelle
2C74         lea     5Ø(a6),aØ           ; aØ -> Eingabepuffer
2C78         move.w  1C(a6),dØ           ; dØ := Pufferzeiger
2C7C         bgt.s   FC2C88              ; Puffer nicht leer: ->
2C7E         move.w  #1,2Ø(a6)           ; Flag 'Puffer leer' setzen
2C84         moveq   #Ø,dØ               ; dØ := Ø
2C86         rts
2C88
```

```
2C88 FC2C88  clr.b    0(a0,d0.w)         ; Endemarke an Eingabe anfügen
2C8C         lea      50(a6),a0          ; a0 -> Eingabepuffer
2C90         bsr      FC2B6A             ; ---> Eingabe in Kommandotabelle suchen
2C94         tst.l    d0                 ; gefunden?
2C96         beq.s    FC2CB0             ; nein: ->
2C98         move.w   #1,20(a6)          ; Flag 'Puffer leer' setzen
2C9E         movea.l  d0,a0              ; a0 := d0 -> Tabelleneintrag
2CA0         move.l   A(a0),-(a7)        ; Routinenadresse auf Stack
2CA4         bsr      FC2AE8             ; ---> Kommando ausführen
2CA8         clr.b    1E(a6)             ; Ausgabeflag löschen
2CAC         addq.l   #4,a7              ; Routinenadresse vom Stack nehmen
2CAE         rts
2CB0
2CB0 FC2CB0  lea      50(a6),a0          ; a0 -> Eingabepuffer
2CB4         lea      8(a6),a1           ; a1 -> Speicher für Zahleneingabe
2CB8         bsr      FC2CFC             ; ---> Zahl aus Eingabe erzeugen
2CBC         move.b   d0,22(a6)          ; Stellenzahl abspeichern
2CC0         bne.s    FC2CD2             ; Nur Hex-Ziffern in der Eingabe: ->
2CC2         move.w   #1,20(a6)          ; Flag 'Puffer leer' setzen
2CC8         lea      20(pc),a0          ; a0 := FC2CEA (Text 'unknown symbol')
2CCC         bsr      FC2278             ; ---> PutString
2CD0         rts
2CD2
2CD2 FC2CD2  tst.w    24(a6)             ; Parameter-Eingabe?
2CD6         bne.s    FC2CE8             ; ja: fertig ->
2CD8         moveq    #FE,d0             ; Für Wort-Ausrichtung
2CDA         and.l    8(a6),d0           ; d0 := eingegebene Zahl
2CDE         move.l   d0,C(a6)           ; als aktuelle Adresse speichern
2CE2         bset.b   #0,1E(a6)          ; Ausgabeflag löschen
2CE8 FC2CE8  rts
2CEA
2CEA FC2CEA  DC.B     0A,'unknown symbol'0A,0,0
2CFC
2CFC ;------ Zahl aus Eingabe generieren
2CFC
2CFC FC2CFC  move.l   d2,-(a7)           ; d2 auf Stack retten
2CFE         moveq    #0,d1              ; d1 := 0  (Ergebnisregister)
2D00         moveq    #FF,d2             ; d2 := -1 (Stellenzähler)
2D02         move.l   d1,d0              ; d0 := 0  (Ziffernregister)
2D04         bra.s    FC2D0E             ; --->
```

```
2D06
2D06 FC2D06  addq.l  #5,d0              ; Addition von 10 erzeugt Wert
2D08         addq.l  #5,d0              ; der Hex-Ziffern A,...,F
2D0A FC2D0A  lsl.l   #4,d1              ; Ergebnisregister mal 16
2D0C         add.l   d0,d1              ; + Wert der nächsten Ziffer
2D0E FC2D0E  addq.l  #1,d2              ; Stellenzähler inkrementieren
2D10         move.b  (a0)+,d0           ; d0 := Zeichen aus Puffer
2D12         beq.s   FC2D3A             ; Endemarke: ->
2D14         subi.b  #30,d0             ; d0 := Ziffernwert für 0,...,9
2D18         blt.s   FC2D38             ; keine Ziffer: ->
2D1A         cmpi.b  #A,d0              ; Wert < 10?
2D1E         blt.s   FC2D0A             ; ja: ok ->
2D20         subi.b  #11,d0             ; d0 := Ziffernwert-10 für A,...,F
2D24         blt.s   FC2D38             ; keine Hex-Ziffer: ->
2D26         cmpi.b  #6,d0              ; Wert-10 < 6?
2D2A         blt.s   FC2D06             ; ja: ok ->
2D2C         subi.b  #20,d0             ; d0 := Ziffernwert-10 für a,...,f
2D30         blt.s   FC2D38             ; keine Hex-Ziffer: ->
2D32         cmpi.b  #6,d0              ; Wert-10 < 6?
2D36         blt.s   FC2D06             ; ja: ok ->
2D38 FC2D38  moveq   #0,d2              ; d2 := 0 als Fehlerflag
2D3A FC2D3A  move.l  d1,(a1)            ; Ergebnisregister abspeichern
2D3C         move.l  d2,d0              ; d0 := Stellenzahl oder 0
2D3E         move.l  (a7)+,d2           ; d2 wiederherstellen
2D40         rts
2D42
2D42 ;------
2D42
2D42         DC.B    ' %lx ',0
2D48
2D48 ;------ Klein-Großbuchstaben-Wandlung
2D48
2D48 FC2D48  cmpi.b  #'a',d0            ; d0 < 'a'?
2D4C         blt.s   FC2D58             ; ja: fertig ->
2D4E         cmpi.b  #'z',d0            ; d0 > 'z'?
2D52         bgt.s   FC2D58             ; ja: fertig ->
2D54         subi.b  #20,d0             ; sonst 32 subtrahieren
2D58 FC2D58  rts
2D5A
2D5A         DC.W    0
```

```
2D5C
2D5C ;---------------------------------   Procure
2D5C
2D5C FC2D5C  addq.w  #1,22(a0)        ; sm_Bids um 1 erhöhen
2D60         bne.s   FC2D6A           ; Semaphore bereits gesetzt: ->
2D62         move.l  a1,10(a0)        ; mp_SigTask -> bidMessage
2D66         moveq   #1,d0            ; Ergebnis TRUE
2D68 FC2D68  rts
2D6A
2D6A FC2D6A  jsr     -16E(a6)         ; ---> PutMsg
2D6E         moveq   #0,d0            ; Ergebnis FALSE
2D70         bra.s   FC2D68           ; --->
2D72
2D72 ;---------------------------------   Vacate
2D72
2D72 FC2D72  clr.l   10(a0)           ; mp_SigTask löschen
2D76         subq.w  #1,22(a0)        ; sm_Bids um 1 vermindern
2D7A         bge.s   FC2D7E           ; Semaphore immer noch gesetzt: ->
2D7C FC2D7C  rts
2D7E
2D7E FC2D7E  move.l  a0,-(a7)         ; a0 retten
2D80         jsr     -174(a6)         ; ---> GetMsg
2D84         movea.l (a7)+,a0         ; a0 wiederherstellen
2D86         move.l  d0,10(a0)        ; mp_SigTask -> Message
2D8A         beq.s   FC2D7C           ; keine Message vorhanden: ->
2D8C         movea.l d0,a1            ; a1 := d0 -> Message
2D8E         jsr     -17A(a6)         ; ---> ReplyMsg
2D92         bra.s   FC2D7C           ; --->
2D94
2D94 ;---------------------------------   InitSemaphore
2D94
2D94 FC2D94  lea     10(a0),a1        ; a1 -> ss_WaitQueue
2D98         move.l  a1,(a1)          ; List Header initialisieren
2D9A         addq.l  #4,(a1)
2D9C         clr.l   4(a1)
2DA0         move.l  a1,8(a1)
2DA4         clr.l   28(a0)           ; ss_Owner löschen
2DA8         clr.w   E(a0)            ; ss_NestCount löschen
2DAC         move.w  #-1,2C(a0)       ; ss_QueueCount initialisieren
2DB2         rts
```

```
2DB4
2DB4 ;--------------------------------  ObtainSemaphore
2DB4
2DB4 FC2DB4  addq.b  #1,127(a6)        ; Forbid
2DB8         addq.w  #1,2C(a0)         ; ss_QueueCount um 1 erhöhen
2DBC         bne.s   FC2DC6            ; war bereits erhöht: ->
2DBE FC2DBE  move.l  114(a6),28(a0)    ; ss_Owner := ThisTask
2DC4         bra.s   FC2DFA            ; --->
2DC6
2DC6 FC2DC6  movem.l d0-d1/a0-a1,-(a7) ; Register retten
2DCA         movea.l 114(a6),a1        ; a1 := ThisTask
2DCE         cmpa.l  28(a0),a1         ; ss_Owner = ThisTask?
2DD2         beq.s   FC2DF6            ; ja: ->
2DD4         lea     -C(a7),a7         ; Platz für List Header im Stack
2DD8         move.l  a1,8(a7)          ; mlh_TailPred := ThisTask
2DDC         bclr.b  #4,1D(a1)         ; sigf_Single löschen
2DE2         lea     10(a0),a0         ; a0 -> ss_WaitQueue
2DE6         movea.l a7,a1             ; a1 -> List Header
2DE8         bsr     FC15E8            ; ---> AddTail
2DEC         moveq   #10,d0            ; sigf_Single Bit setzen
2DEE         jsr     -13E(a6)          ; ---> Wait
2DF2         lea     C(a7),a7          ; List Header wieder entfernen
2DF6 FC2DF6  movem.l (a7)+,d0-d1/a0-a1 ; Register wiederherstellen
2DFA FC2DFA  addq.w  #1,E(a0)          ; ss_NestCount um 1 erhöhen
2DFE         jsr     -8A(a6)           ; ---> Permit
2E02         rts
2E04
2E04 ;--------------------------------  ReleaseSemaphore
2E04
2E04 FC2E04  subq.w  #1,E(a0)          ; ss_NestCount um 1 erniedrigen
2E08         beq.s   FC2E12            ; Semaphore frei: ->
2E0A         bmi.s   FC2E50            ; einmal zu viel: Fehler ->
2E0C         subq.w  #1,2C(a0)         ; ss_QueueCount um 1 erniedrigen
2E10         bra.s   FC2E4E            ; --->
2E12
2E12 FC2E12  addq.b  #1,127(a6)        ; Forbid
2E16         subq.w  #1,2C(a0)         ; ss_QueueCount um 1 erniedrigen
2E1A         blt.s   FC2E46            ; keine wartende Task: ->
2E1C         movem.l d0-d1/a1,-(a7)    ; Register retten
2E20         move.l  a0,d1             ; a0 in d1 retten
```

```
2E22         lea      10(a0),a0          ; a0 -> ss_WaitQueue
2E26         bsr      FC160E             ; ---> RemHead
2E2A         tst.l    d0                 ; War Liste leer?
2E2C         beq.s    FC2E50             ; ja: Fehler ->
2E2E         movea.l d1,a0               ; a0 wiederherstellen
2E30         movea.l d0,a1               ; a1 -> 1. Node
2E32         movea.l 8(a1),a1            ; a1 -> ssr_Waiter
2E36         move.l   a1,28(a0)          ; ss_Owner := ssr_Waiter
2E3A         moveq    #10,d0             ; d0 := sigf_Single
2E3C         jsr      -144(a6)           ; ---> Signal
2E40         movem.l (a7)+,d0-d1/a1      ; Register wiederherstellen
2E44         bra.s    FC2E4A             ; --->
2E46
2E46 FC2E46  clr.l    28(a0)             ; ss_Owner löschen
2E4A FC2E4A  jsr      -8A(a6)            ; ---> Permit
2E4E FC2E4E  rts
2E50
2E50 ;------ Semaphore-Verschachtelungsfehler
2E50
2E50 FC2E50  movem.l d7/a5-a6,-(a7)      ; Register retten
2E54         move.l   #81000008,d7       ; d7 := Alert-Kode 'AN_SemCorrupt'
2E5A         movea.l 4,a6                ; a6 := SysBase
2E5E         jsr      -6C(a6)            ; ---> Alert
2E62         movem.l (a7)+,d7/a5-a6      ; Register wiederherstellen
2E66         bra.s    FC2E4E             ; --->
2E68
2E68 ;----------------------------------  AttemptSemaphore
2E68
2E68 FC2E68  movea.l 114(a6),a1          ; a1 := ThisTask
2E6C         addq.b   #1,127(a6)         ; Forbid
2E70         addq.w   #1,2C(a0)          ; ss_QueueCount inkrementieren
2E74         beq.s    FC2E88             ; Warteschlange leer: ->
2E76         cmpa.l   28(a0),a1          ; ss_Owner = ThisTask?
2E7A         beq.s    FC2E8C             ; ja: ->
2E7C         subq.w   #1,2C(a0)          ; ss_QueueCount dekrementieren
2E80         jsr      -8A(a6)            ; ---> Permit
2E84         moveq    #0,d0              ; Ergebnis FALSE
2E86         bra.s    FC2E96             ; --->
2E88
2E88 FC2E88  move.l   a1,28(a0)          ; ss_Owner := ThisTask
```

```
2E8C FC2E8C  addq.w  #1,E(a0)        ; ss_NestCount inkrementieren
2E90         jsr     -8A(a6)         ; ---> Permit
2E94         moveq   #1,d0           ; Ergebnis TRUE
2E96 FC2E96  rts
2E98
2E98 ;---------------------------------  ObtainSemaphoreList
2E98
2E98 FC2E98  movem.l d2/a2-a3,-(a7)  ; Register retten
2E9C         moveq   #0,d1
2E9E         movea.l 114(a6),a2      ; a2 := ThisTask
2EA2         addq.b  #1,127(a6)      ; Forbid
2EA6         movea.l a0,a3           ; a3 := a0 -> Semaphore List
2EA8         move.l  0(a3),d2        ; d2 := lh_Head
2EAC FC2EAC  movea.l d2,a1           ; a1 -> nächster Node
2EAE         move.l  (a1),d2         ; d2 := ln_Succ
2EB0         beq.s   FC2EDC          ; Ende der Liste: ->
2EB2         addq.w  #1,2C(a1)       ; ss_QueueCount inkrementieren
2EB6         beq.s   FC2ED2          ; Warteschlange war leer: ->
2EB8         cmpa.l  28(a1),a2       ; ss_Owner = ThisTask?
2EBC         beq.s   FC2ED6          ; ja: ->
2EBE         move.l  a2,24(a1)       ; ssr_Waiter := ThisTask
2EC2         lea     10(a1),a0       ; a0 -> ss_WaitQueue
2EC6         lea     1C(a1),a1       ; a1 -> ss_MultipleLink
2ECA         bsr     FC15E8          ; ---> AddTail
2ECE         moveq   #1,d1           ; d1 := 1
2ED0         bra.s   FC2EAC          ; ---> Loop
2ED2
2ED2 FC2ED2  move.l  a2,28(a1)       ; ss_Owner := ThisTask
2ED6 FC2ED6  addq.w  #1,E(a1)        ; ss_NextCount inkrementieren
2EDA         bra.s   FC2EAC          ; ---> Loop
2EDC
2EDC FC2EDC  tst.l   d1              ; Ist Task in einer Warteschlange?
2EDE         beq.s   FC2F04          ; nein: ->
2EE0         move.l  0(a3),d2        ; d2 -> 1. Node der Semaphore List
2EE4 FC2EE4  movea.l d2,a3           ; a3 -> nächster Node
2EE6         move.l  (a3),d2         ; d2 -> ln_Succ
2EE8         beq.s   FC2F04          ; Ende der Liste: ->
2EEA FC2EEA  cmpa.l  28(a3),a2       ; ss_Owner = ThisTask?
2EEE         bne.s   FC2EFC          ; nein: ->
2EF0         tst.w   E(a3)           ; ss_NestCount = 0?
```

```
2EF4         bne.s    FC2EE4           ; nein: ->
2EF6         addq.w   #1,E(a3)         ; ss_NestCount inkrementieren
2EFA         bra.s    FC2EE4           ; ---> Loop
2EFC
2EFC FC2EFC  moveq    #10,d0           ; d0 := sigf_Single
2EFE         jsr      -13E(a6)         ; ---> Wait
2F02         bra.s    FC2EEA           ; ---> Loop
2F04
2F04 FC2F04  jsr      -8A(a6)          ; ---> Permit
2F08         movem.l (a7)+,d2/a2-a3    ; Register wiederherstellen
2F0C         rts
2F0E
2F0E ;--------------------------------  ReleaseSemaphoreList
2F0E
2F0E FC2F0E  move.l  d2,-(a7)          ; d2 auf Stack retten
2F10         move.l  0(a0),d2          ; d2 -> 1. Node der Semaphore Liste
2F14 FC2F14  movea.l d2,a0             ; a0 -> nächster Node
2F16         move.l  (a0),d2           ; d2 := ln_Succ
2F18         beq.s   FC2F20            ; Ende der Liste: ->
2F1A         jsr     -23A(a6)          ; ---> ReleaseSemaphore
2F1E         bra.s   FC2F14            ; ---> Loop
2F20
2F20 FC2F20  move.l  (a7)+,d2          ; d2 wiederherstellen
2F22         rts
2F24
2F24 ;--------------------------------  AddSemaphore
2F24
2F24 FC2F24  jsr     -22E(a6)          ; ---> InitSemaphore
2F28         lea     214(a6),a0        ; a0 -> Semaphore List
2F2C         bra     FC1682            ; ---> Forbid, Enqueue, Permit
2F30
2F30 ;--------------------------------  RemSemaphore
2F30
2F30 FC2F30  bra     FC168E            ; ---> Forbid, Remove, Permit
2F34
2F34 ;--------------------------------  FindSemaphore
2F34
2F34 FC2F34  lea     214(a6),a0        ; a0 -> Semaphore List
2F38         jsr     -114(a6)          ; ---> FindName
2F3C         rts
```

```
2F3E
2F3E         DC.W    Ø
2F4Ø
2F4Ø ;--------------------------------  CopyMemQuick
2F4Ø
2F4Ø FC2F4Ø  moveq   #Ø,d1
2F42         bra.s   FC2F64
2F44
2F44 ;--------------------------------  CopyMem
2F44
2F44 FC2F44  moveq   #C,d1               ; d1 := 12
2F46         cmp.l   d1,dØ               ; Bereichslänge < 12?
2F48         bcs.s   FC2FA2              ; ja: byteweise kopieren ->
2F4A         move.l  aØ,d1               ; d1 := aØ -> Quellbereich
2F4C         btst.l  #Ø,d1               ; ungerade Adresse?
2F5Ø         beq.s   FC2F56              ; nein: ->
2F52         move.b  (aØ)+,(a1)+         ; 1 Byte kopieren
2F54         subq.l  #1,dØ               ; Bereichslänge dekrementieren
2F56 FC2F56  move.l  a1,d1               ; d1 := a1 -> Zielbereich
2F58         btst.l  #Ø,d1               ; ungerade Adresse?
2F5C         bne.s   FC2FA2              ; ja: byteweise kopieren ->
2F5E         move.l  dØ,d1               ; d1 := dØ = Bereichslänge
2F6Ø         andi.w  #3,d1               ; d1 := Bereichslänge modulo 4
2F64 FC2F64  move.w  d1,-(a7)            ; d1 auf Stack retten
2F66         moveq   #6Ø,d1              ; d1 := 96
2F68         cmp.l   d1,dØ               ; Bereichslänge < 96?
2F6A         bcs.s   FC2F86              ; ja: ->
2F6C         movem.l d1-d7/a2-a6,-(a7)   ; Register auf Stack retten
2F7Ø FC2F7Ø  movem.l (aØ)+,d1-d7/a2-a6   ; 12 Langworte aus dem Quellbereich
2F74         movem.l d1-d7/a2-a6,(a1)    ; über Register in Zielbereich kopieren
2F78         moveq   #3Ø,d1              ; d1 := Länge des kopierten Bereichs
2F7A         adda.l  d1,a1               ; Zielzeiger um d1 erhöhen
2F7C         sub.l   d1,dØ               ; Kopierlänge von Bereichslänge subtr.
2F7E         cmp.l   d1,dØ               ; noch mind. eine Kopierlänge übrig?
2F8Ø         bcc.s   FC2F7Ø              ; ja: Kopiervorgang wiederholen ->
2F82         movem.l (a7)+,d1-d7/a2-a6   ; Register wiederherstellen
2F86 FC2F86  lsr.l   #2,dØ               ; dØ := Restlänge in Langworten
2F88         beq.s   FC2F9A              ; = Ø: ->
2F8A         subq.l  #1,dØ               ; dØ := Anzahl der Langworte-1
2F8C         move.l  dØ,d1               ; d1 := Kopierschleifenzähler
```

```
2F8E         swap    dØ                   ; dØ := H-Wort der Anzahl
2F9Ø FC2F9Ø  move.l  (aØ)+,(a1)+          ; Kopierschleife für Langworte
2F92         dbra    d1,FC2F9Ø
2F96         dbra    dØ,FC2F9Ø
2F9A FC2F9A  move.w  (a7)+,d1             ; d1 := Zahl der restlichen Bytes
2F9C         beq.s   FC2FB2               ; = Ø: ->
2F9E         moveq   #Ø,dØ                ; dØ := Ø
2FAØ         bra.s   FC2FAA               ; --->
2FA2
2FA2 FC2FA2  move.w  dØ,d1                ; d1 := Kopierschleifenzähler
2FA4         swap    dØ                   ; dØ := H-Wort der Anzahl
2FA6         bra.s   FC2FAA               ; --->
2FA8
2FA8 FC2FA8  move.b  (aØ)+,(a1)+          ; Kopierschleife für Bytes
2FAA FC2FAA  dbra    d1,FC2FA8
2FAE         dbra    dØ,FC2FA8
2FB2 FC2FB2  rts
2FB4
2FB4 ;------ Default TaskTrap und TaskException
2FB4
2FB4 FC2FB4  movem.l dØ-d7/aØ-a7,18Ø      ; Register retten
2FBA         lea     2(a7),a5             ; a5 -> PC auf Stack
2FBE         move.l  4,dØ                 ; dØ := SysBase
2FC2         btst.l  #Ø,dØ                ; dØ ungerade?
2FC6         bne.s   FC2FCE               ; ja: ->
2FC8         movea.l dØ,aØ                ; aØ := dØ = SysBase
2FCA         lea     114(aØ),a5           ; a5 -> ThisTask
2FCE FC2FCE  move.l  (a7),d7              ; d7 := 1. Langwort auf Stack
2FDØ         andi.l  #FFFF,d7             ; d7 := Nr. des Ausnahmevektors
2FD6
2FD6 ;---------------------------------  Alert
2FD6
2FD6 FC2FD6  move.w  #4ØØØ,INTENA         ; Interrupts sperren
2FDE         move.l  #'HELP',dØ
2FE4         cmp.l   Ø,dØ                 ; steht schon 'HELP' ab Adresse Ø?
2FE8         beq     FC3Ø5E               ; ja: Alert im Alert ->
2FEC         move.l  dØ,Ø                 ; 'HELP' eintragen
2FFØ         lea     1ØØ,aØ               ; aØ -> Zwischenspeicher
2FF4         move.l  d7,(aØ)+             ; 1. Eintrag: Alert-Kode/Vektornummer
2FF6         move.l  (a5),(aØ)+           ; 2. Eintrag: ThisTask
```

```
2FF8         move.l  4,d0              ; d0 := SysBase
2FFC         move.l  d0,d1             ; d1 := d0
2FFE         andi.l  #FF0001,d1        ; wirklich SysBase?
3004         bne.s   FC305E            ; nein: ->
3006         movea.l d0,a6             ; a6 := d0 = SysBase
3008         add.l   26(a6),d0         ; + ChkBase
300C         addq.l  #1,d0             ; + 1 sollte = 0 sein!
300E         bne.s   FC305E            ; ist es aber nicht: ->
3010         move.l  #F1E2D3C4,d0      ; d0 mit Bitmuster belegen
3016         move.l  d0,-(a7)          ; und auf Stack legen
3018         cmp.l   (a7)+,d0          ; liegt es dort?
301A         bne.s   FC3054            ; nein: kein Stack ->
301C         tst.l   d7                ; HÖchstes Bit in d7 gesetzt?
301E         bmi.s   FC305E            ; ja: ->
3020         lea     202(a6),a0        ; a0 -> LastAlert
3024         move.l  d7,(a0)+          ; Alert-Kode abspeichern
3026         move.l  (a5),(a0)+        ; ThisTask abspeichern
3028         bsr     FC30EC            ; ---> Alertmeldung ausgeben
302C         andi.l  #FFFF0000,d7      ; Unteres Wort von d7 löschen
3032         bne.s   FC3044            ; Ergebnis nicht Null: ->
3034         tst.l   d0                ; Linker Mausknopf gedrückt?
3036         movem.l 180,d0-d7/a0-a7   ; Register wiederherstellen
303C         bne     FC05F0            ; ja: reboot ->
3040         bra     FC2342            ; ---> Debug
3044
3044 FC3044  tst.l   126(a6)           ; Interrupts gesperrt?
3048         bge.s   FC3052            ; ja: ->
304A         move.w  #C000,INTENA      ; Interrupts freigeben
3052 FC3052  rts
3054
3054 FC3054  movea.l #40000,a7         ; Stack neu initialisieren
305A         clr.l   -(a7)             ; Platz schaffen für PC
305C         clr.w   -(a7)             ; und SR
305E FC305E  ori.b   #3,BFE201         ; 8520-A DDRA: Bits 0 und 1 Output
3066         andi.b  #FE,BFE001        ; 8520-A PA0 löschen (OVL)
306E         move.l  #FC3076,20        ; Vektor für Privilegverletzung setzen
3076 FC3076  move    #2700,sr          ; IR-Ebene 7 setzen
307A         moveq   #5,d1             ; Zähler für Blinkschleife setzen
307C         move.w  #174,SERPER       ; Baudrate auf 9600 setzen
3084 FC3084  moveq   #FF,d0            ; Zähler für Blinkphase setzen
```

```
3086 FC3086  bset.b  #1,BFE001           ; LED dunkel
308E         dbra    d0,FC3086
3092 FC3092  bclr.b  #1,BFE001           ; LED hell
309A         dbra    d0,FC3092
309E         move.w  SERDATR,d0          ; Serielles Datenregister lesen
30A4         move.w  #800,INTREQ         ; Bit 'Receive Buffer full' löschen
30AC         andi.b  #7F,d0              ; Statusbits löschen
30B0         cmpi.b  #7F,d0              ; Kode 127 empfangen?
30B4         dbeq    d1,FC3084           ; nein: bis zu 5 mal wiederholen ->
30B8         bmi     FC05F0              ; dann: reboot ->
30BC         move.l  d7,-(a7)
30BE         jmp     FC2342              ; ---> Debug
30C4
30C4 FC30C4  move.l  #-1,d6              ; Für LastAlert
30CA         cmpi.l  #'HELP',0           ; Alert im Alert?
30D2         bne     FC014C              ; nein: Exec-Initialisierung ->
30D6         clr.l   0                   ; 'HELP' löschen
30DA         movem.l 100,d6-d7           ; Für LastAlert
30E0         bra     FC014C              ; ---> Exec-Initialisierung
30E4
30E4 FC30E4  movem.l d6-d7,202(a6)       ; LastAlert-Werte speichern
30EA         rts
30EC
30EC ;------ Alert-Meldung ausgeben
30EC
30EC FC30EC  movem.l d2/d7/a2-a3/a6,-(a7) ; Register retten
30F0         moveq   #A,d1               ; H-Zählerwert für Warteschleife
30F2         moveq   #FF,d0              ; L-Zählerwert für Warteschleife
30F4 FC30F4  dbra    d0,FC30F4           ; Warteschleife
30F8         dbra    d1,FC30F4
30FC         move.l  202(a6),d2          ; d2 := Alert-Kode oder -1
3100         moveq   #FF,d0              ; d0 := -1
3102         cmp.l   d0,d2               ; d2 = -1?
3104         beq.s   FC317C              ; ja: ->
3106         lea     -C8(a7),a7          ; 200-Byte-Ausgabepuffer anlegen
310A         lea     (a7),a3             ; a3 -> Ausgabepuffer
310C         lea     9C(pc),a0           ; a0 := FC31AA 'Software Failure'
3110         move.l  d2,d0               ; d0 := d2
3112         swap    d0                  ; d0.w := oberes Wort von d2
3114         cmpi.b  #1,d0               ; d0.b = 1?
```

```
3118          bne.s    FC3120             ; nein: ->
311A          lea      78(pc),a0          ; a0 := FC3194 'Not enough memory'
311E          bra.s    FC312E             ; --->
3120
3120 FC3120   btst.l   #1F,d2             ; Höchstes Bit von d2 gesetzt?
3124          bne.s    FC312E             ; ja: ->
3126          tst.w    d0                 ; d2 = Vektornummer?
3128          beq.s    FC312E             ; ja: ->
312A          lea      93(pc),a0          ; a0 := FC31BF 'Recoverable Alert.'
312E FC312E   bsr.s    FC318A             ; ---> Text in Ausgabepuffer
3130          lea      A3(pc),a0          ; a0 := FC31D5 'Press left mouse ...'
3134          bsr.s    FC318A             ; ---> Text in Ausgabepuffer
3136          clr.b    (a3)+              ; Endekode anfügen
3138          lea      C2(pc),a0          ; a0 := FC31FC 'Guru Meditation ...'
313C          lea      202(a6),a1         ; a1 -> Ausgabe-Daten
3140          lea      42(pc),a2          ; a2 := FC3184 (Ausgaberoutine)
3144          jsr      -20A(a6)           ; ---> RawDoFmt
3148          lea      D1(pc),a1          ; a1 := FC321B 'intuition.library'
314C          moveq    #0,d0              ; Version beliebig
314E          jsr      -228(a6)           ; ---> OpenLibrary
3152          tst.l    d0                 ; Library geöffnet?
3154          beq.s    FC316E             ; nein: ->
3156          movea.l  a6,a3              ; a6 in a3 retten
3158          movea.l  d0,a6              ; a6 := IntuitionBase
315A          clr.l    d0                 ; Alertnummer := 0
315C          lea      (a7),a0            ; a0 -> Ausgabepuffer
315E          moveq    #28,d1             ; d1 := 40 = Ausgabehöhe
3160          jsr      -5A(a6)            ; ---> DisplayAlert
3164          movea.l  d0,a2              ; d0 in a2 retten
3166          movea.l  a6,a1              ; a1 := a6 = IntuitionBase
3168          movea.l  a3,a6              ; a6 := a3 = SysBase
316A          jsr      -19E(a6)           ; ---> CloseLibrary
316E FC316E   lea      C8(a7),a7          ; Ausgabepuffer auflösen
3172          clr.l    0                  ; 'HELP' löschen
3176          moveq    #FF,d0             ; d0 := -1
3178          move.l   d0,202(a6)         ; als LastAlert speichern
317C FC317C   move.l   a2,d0              ; d0 := Alert-Resultat
317E          movem.l  (a7)+,d2/d7/a2-a3/a6 ; Register wiederherstellen
3182          rts
3184
```

```
3184 ;------ Ausgaberoutine für RawDoFmt
3184
3184 FC3184  move.b  dØ,(a3)+          ; Zeichen in Ausgabepuffer
3186         clr.b   (a3)              ; Endekode anfügen
3188         rts
318A
318A ;------ Text in Ausgabepuffer kopieren
318A
318A FC318A  clr.b   (a3)+             ; Kode Ø in Puffer
318C FC318C  move.b  (aØ)+,(a3)+       ; String in Puffer kopieren
318E         bne.s   FC318C            ; bis Endekode
319Ø         st      (a3)+             ; Fortsetzungskode dahinter
3192         rts
3194
3194 ;------ Alert-Texte
3194
3194 FC3194  DC.B    26,ØF,'Not enough memory. ',Ø
31AA FC31AA  DC.B    26,ØF,'Software Failure. ',Ø
31BF FC31BF  DC.B    26,ØF,'Recoverable Alert. ',Ø
31D5 FC31D5  DC.B    EA,ØF,'Press left mouse button to continue.',Ø
31FC FC31FC  DC.B    8E,1E,'Guru Meditation #%Ø8lx.%Ø8lx',Ø
321B
321B FC321B  DC.B    'intuition.library',Ø
322D
322D ;------ Alert Resident-Struktur
322D
322D FC322D  DC.B    'alert.hook',ØD,ØA,Ø
323A
323A FC323A  DC.W    4AFC              ; rt_MatchWord
323C         DC.L    FC323A            ; rt_MatchTag
324Ø         DC.L    FC3254            ; rt_Endskip
3244         DC.B    1                 ; rt_Flags
3245         DC.B    21                ; rt_Version
3246         DC.B    Ø                 ; rt_Type
3247         DC.B    5                 ; rt_Pri
3248         DC.L    FC322D            ; rt_Name
324C         DC.L    FC322D            ; rt_IdString
325Ø         DC.L    FC3ØEC            ; rt_Init
3254
3254 ;------ Tabelle für Debug-Tastenkommandokos
```

```
3254
3254 FC3254  DC.L    FC325E              ; Zeiger zum nächsten Tabelleneintrag
3258         DC.B    4,Ø                 ; CTRL-D
325A         DC.L    FC2442
325E FC325E  DC.L    FC3268
3262         DC.B    D,Ø                 ; RETURN
3264         DC.L    FC251Ø
3268 FC3268  DC.L    FC3272
326C         DC.B    9,Ø                 ; TAB
326E         DC.L    FC2426
3272 FC3272  DC.L    FC327C
3276         DC.B    '?',Ø
3278         DC.L    FC2BØ6
327C FC327C  DC.L    FC3286
328Ø         DC.B    '.',Ø
3282         DC.L    FC2562
3286 FC3286  DC.L    FC329Ø
328A         DC.B    ',',Ø
328C         DC.L    FC2572
329Ø FC329Ø  DC.L    FC329A
3294         DC.B    '>',Ø
3296         DC.L    FC2522
329A FC329A  DC.L    FC32A4
329E         DC.B    '<',Ø
32AØ         DC.L    FC2542
32A4 FC32A4  DC.L    FC32AE
32A8         DC.B    8,Ø                 ; BACKSPACE
32AA         DC.L    FC2542
32AE FC32AE  DC.L    FC32B8
32B2         DC.B    ' ',Ø
32B4         DC.L    FC2522
32B8 FC32B8  DC.L    FC32C2
32BC         DC.B    5B,Ø                ; eckige Klammer auf
32BE         DC.L    FC2582
32C2 FC32C2  DC.L    FC32CC
32C6         DC.B    5D,Ø                ; eckige Klammer zu
32C8         DC.L    FC25AØ
32CC FC32CC  DC.L    FC32D6
32DØ         DC.B    ':',Ø
32D2         DC.L    FC26Ø2
```

```
32D6 FC32D6  DC.L    FC32E0
32DA         DC.B    '+',0
32DC         DC.L    FC25B4
32E0 FC32E0  DC.L    FC32EA
32E4         DC.B    '-',0
32E6         DC.L    FC25D4
32EA FC32EA  DC.L    FC32F4
32EE         DC.B    '=',0
32F0         DC.L    FC2642
32F4 FC32F4  DC.L    FC32FE
32F8         DC.B    '!',0
32FA         DC.L    FC2938
32FE FC32FE  DC.L    FC3308
3302         DC.B    '^',0
3304         DC.L    FC29B2
3308 FC3308  DC.L    FC3312
330C         DC.B    '_',0
330E         DC.L    FC2BA0
3312 FC3312  DC.L    FC331C
3316         DC.B    '09'
3318         DC.L    FC2BA0
331C FC331C  DC.L    FC3326
3320         DC.B    'az'
3322         DC.L    FC2BA0
3326 FC3326  DC.L    FC3330
332A         DC.B    'AZ'
332C         DC.L    FC2BA0
3330 FC3330  DC.L    0
3334
3334 ;------ Tabelle für Debug-Parametereingabe
3334
3334 FC3334  DC.L    FC333E
3338         DC.B    8,0                   ; BACKSPACE
333A         DC.L    FC2C44
333E FC333E  DC.L    FC3348
3342         DC.B    D,0                   ; RETURN
3344         DC.L    FC2C6E
3348 FC3348  DC.L    FC3352
334C         DC.B    18,0                  ; CTRL-X
334E         DC.L    FC2C2C
```

```
3352 FC3352  DC.L    FC335C
3356         DC.B    15,Ø               ; CTRL-U
3358         DC.L    FC2C2C
335C FC335C  DC.L    FC3366
336Ø         DC.B    '>',Ø
3362         DC.L    FC2522
3366 FC3366  DC.L    FC337Ø
336A         DC.B    '<',Ø
336C         DC.L    FC2542
337Ø FC337Ø  DC.L    FC337A
3374         DC.B    ' ',Ø
3376         DC.L    FC2C2A
337A FC337A  DC.L    FC3384
337E         DC.B    '_',Ø
338Ø         DC.L    FC2BAØ
3384 FC3384  DC.L    FC338E
3388         DC.B    'Ø9'
338A         DC.L    FC2BAØ
338E FC338E  DC.L    FC3398
3392         DC.B    'az'
3394         DC.L    FC2BAØ
3398 FC3398  DC.L    FC33A2
339C         DC.B    'AZ'
339E         DC.L    FC2BAØ
33A2 FC33A2  DC.L    Ø
33A6
33A6 ;------ Tabelle der Debug-Kommandonamen
33A6
33A6 FC33A6  DC.B    'alter',Ø
33AC FC33AC  DC.B    'boot',Ø
33B1 FC33B1  DC.B    'clear',Ø
33B7 FC33B7  DC.B    'fill',Ø
33BC FC33BC  DC.B    'find',Ø
33C1 FC33C1  DC.B    'go',Ø
33C4 FC33C4  DC.B    'ig',Ø
33C7 FC33C7  DC.B    'limit',Ø
33CD FC33CD  DC.B    'list',Ø
33D2 FC33D2  DC.B    'regs',Ø
33D7 FC33D7  DC.B    'reset',Ø
33DD FC33DD  DC.B    'resume',Ø
```

```
33E4 FC33E4  DC.B    'set',Ø
33E8 FC33E8  DC.B    'show',Ø
33ED FC33ED  DC.B    'user',Ø
33F2         DC.W    Ø
33F4
33F4 ;------ Tabelle der Routinenadressen zu den Kommandonamen
33F4
33F4 FC33F4  DC.L    FC34Ø2              ; Zeiger auf nächsten Eintrag
33F8         DC.L    FC33A6              ; 'alter'
33FC         DC.W    1
33FE         DC.L    FC267Ø              ; Adresse der zugehörigen Routine
34Ø2 FC34Ø2  DC.L    FC341Ø
34Ø6         DC.L    FC33AC              ; 'boot'
34ØA         DC.W    1
34ØC         DC.L    FCØ5FØ
341Ø FC341Ø  DC.L    FC341E
3414         DC.L    FC33B1              ; 'clear'
3418         DC.W    1
341A         DC.L    FC28A2
341E FC341E  DC.L    FC342C
3422         DC.L    FC33B7              ; 'fill'
3426         DC.W    1
3428         DC.L    FC2A38
342C FC342C  DC.L    FC343A
343Ø         DC.L    FC33BC              ; 'find'
3434         DC.W    1
3436         DC.L    FC29CØ
343A FC343A  DC.L    FC3448
343E         DC.L    FC33C1              ; 'go'
3442         DC.W    1
3444         DC.L    FC2438
3448 FC3448  DC.L    FC3456
344C         DC.L    FC33C4              ; 'ig'
345Ø         DC.W    1
3452         DC.L    FCØ5FØ
3456 FC3456  DC.L    FC3464
345A         DC.L    FC33C7              ; 'limit'
345E         DC.W    1
346Ø         DC.L    FC29B2
3464 FC3464  DC.L    FC3472
```

```
3468        DC.L    FC33CD          ; 'list'
346C        DC.W    1
346E        DC.L    FC26B2
3472 FC3472 DC.L    FC3480
3476        DC.L    FC33D2          ; 'regs'
347A        DC.W    1
347C        DC.L    FC27E6
3480 FC3480 DC.L    FC348E
3484        DC.L    FC33D7          ; 'reset'
3488        DC.W    1
348A        DC.L    FC28B2
348E FC348E DC.L    FC349C
3492        DC.L    FC33DD          ; 'resume'
3496        DC.W    1
3498        DC.L    FC2442
349C FC349C DC.L    FC34AA
34A0        DC.L    FC33E4          ; 'set'
34A4        DC.W    1
34A6        DC.L    FC28D0
34AA FC34AA DC.L    FC34B8
34AE        DC.L    FC33E8          ; 'show'
34B2        DC.W    1
34B4        DC.L    FC290A
34B8 FC34B8 DC.L    FC34C6
34BC        DC.L    FC33ED          ; 'user'
34C0        DC.W    1
34C2        DC.L    FC24B2
34C6 FC34C6 DC.L    0
34CA
34CA ;************************ Ende von Exec ********************************
```

DOS - Bootstrap

```
8884 ************************************************************************
8884 *                                                                      *
8884 *  D O S - B O O T S T R A P                                           *
8884 *                                                                      *
8884 ************************************************************************
8884
8884 FE8884  DC.W    4AFC              ; rt_Matchword
8886         DC.L    FE8884            ; rt_Matchtag
888A         DC.L    FE88C0            ; rt_Endskip
888E         DC.B    1                 ; rt_Flags
888F         DC.B    21                ; rt_Version
8890         DC.B    0                 ; rt_Type
8891         DC.B    C4                ; rt_Pri
8892         DC.L    FE889E            ; rt_Name
8896         DC.L    FE88A4            ; rt_IdString
889A         DC.L    FE88D6            ; rt_Init
889E
889E FE889E  DC.B    'strap',0
88A4 FE88A4  DC.B    'strap 33.97 (1 Oct 1986)',0D,0A,0,0
88C0
88C0 FE88C0  DC.B    'DOS',0
88C4 FE88C4  DC.B    'trackdisk.device',0,0
88D6
88D6 FE88D6  movem.l d2-d3/a3-a5,-(a7) ; Register retten
88DA         moveq   #0,d3             ; d3 := 0
88DC         suba.l  a4,a4             ; a4 := 0
88DE         lea     FE8B3A,a3         ; a3 -> rts
88E4         link    a5,#-7E           ; Platz für 126 Bytes im Stack
88E8         suba.l  #7E,a5            ; a5 auf Stack-Bereich setzen
88EE         move.l  a6,0(a5)          ; SysBase abspeichern
88F2         move.l  d3,4(a5)          ; Platz für GfxBase löschen
```

```
88F6         move.l  #488,d0          ; Pufferlänge 1160 Bytes
88FC         move.l  #10002,d1        ; Speichertyp CHIP, CLEAR
8902         jsr     -C6(a6)          ; ---> AllocMem
8906         tst.l   d0               ; Speicher reserviert?
8908         bne.s   FE8924           ; ja: ->
890A         movem.l d7/a5-a6,-(a7)   ; Register retten
890E         move.l  #30010000,d7     ; d7 := Alert-Kode
8914         movea.l 4,a6             ; a6 := SysBase
8918         jsr     -6C(a6)          ; ---> Alert
891C         movem.l (a7)+,d7/a5-a6   ; Register wiederherstellen
8920         bra     FE8B2A           ; ---> Abschluß
8924
8924 FE8924  movea.l d0,a4            ; a4 := d0 -> Puffer
8926         lea     FE889E,a0        ; a0 -> rt_Name 'strap'
892C         move.l  a0,36(a5)        ; in IOStdReq und
8930         move.l  a0,66(a5)        ; in ReplyPort eintragen
8934         suba.l  a1,a1            ; a1 := 0
8936         jsr     -126(a6)         ; ---> FindTask
893A         move.l  d0,6C(a5)        ; SigTask in ReplyPort eintragen
893E         move.b  #0,6A(a5)        ; Flags löschen
8944         lea     70(a5),a0        ; a0 -> mp_MsgList Header
8948         move.l  a0,(a0)          ; Header initialisieren
894A         addq.l  #4,(a0)
894C         clr.l   4(a0)
8950         move.l  a0,8(a0)
8954         moveq   #FF,d0           ; keine bestimmte Signalnummer
8956         jsr     -14A(a6)         ; ---> AllocSignal
895A         move.b  d0,6B(a5)        ; Alloziertes Signal abspeichern
895E         bpl.s   FE897A           ; Signal erfolgreich alloziert: ->
8960         movem.l d7/a5-a6,-(a7)   ; Register retten
8964         move.l  #30070000,d7     ; d7 := Alert-Kode
896A         movea.l 4,a6             ; a6 := SysBase
896E         jsr     -6C(a6)          ; ---> Alert
8972         movem.l (a7)+,d7/a5-a6   ; Register wiederherstellen
8976         bra     FE8B1E           ; ---> Abschluß
897A
897A FE897A  lea     5C(a5),a0        ; a0 -> ReplyPort
897E         move.l  a0,3A(a5)        ; in IOStdReq eintragen
8982         lea     -C0(pc),a0       ; a0 := FE88C4 -> 'trackdisk.device'
8986         lea     2C(a5),a1        ; a1 -> IORequest
```

```
898A          moveq    #0,d0          ; d0 = Unit-Nummer := 0
898C          moveq    #0,d1          ; d1 = Flags := 0
898E          jsr      -1BC(a6)       ; ---> OpenDevice
8992          tst.l    d0             ; erfolgreich?
8994          beq.s    FE89B0         ; ja: ->
8996          movem.l d7/a5-a6,-(a7)  ; Register retten
899A          move.l   #30048014,d7   ; d7 := Alert-Kode
89A0          movea.l 4,a6            ; a6 := SysBase
89A4          jsr      -6C(a6)        ; ---> Alert
89A8          movem.l (a7)+,d7/a5-a6  ; Register wiederherstellen
89AC          bra      FE8B14         ; ---> Abschluß
89B0
89B0 FE89B0   move.w   #100,DMACON    ; Bit Plane DMA sperren
89B8          lea      2C(a5),a1      ; a1 -> IORequest
89BC          move.w   #5,1C(a1)      ; io_Command := cmd_Clear
89C2          jsr      -1C8(a6)       ; ---> DoIO
89C6          tst.l    d0             ; Fehler?
89C8          bne      FE8AC8         ; ja: ->
89CC          lea      2C(a5),a1      ; a1 -> IORequest
89D0          move.w   #D,1C(a1)      ; io_Command := td_Changenum
89D6          jsr      -1C8(a6)       ; ---> DoIO
89DA          tst.l    d0             ; Disk gewechselt?
89DC          bne      FE8AC8         ; ja: ->
89E0          move.l   4C(a5),d2      ; d2 := io_Actual
89E4          lea      2C(a5),a1      ; a1 -> IORequest
89E8          move.w   #2,1C(a1)      ; io_Command := cmd_Read
89EE          move.l   #400,24(a1)    ; io_Length := 1024 Bytes
89F6          move.l   a4,28(a1)      ; io_Data := a4
89FA          move.l   #0,2C(a1)      ; io_Offset := 0
8A02          jsr      -1C8(a6)       ; ---> DoIO
8A06          tst.l    d0             ; Fehler?
8A08          bne.s    FE8A5C         ; ja: ->
8A0A          move.l   (a4),d0        ; d0 := 1. gelesenes Langwort
8A0C          cmp.l    -14E(pc),d0    ; d0 = 'DOS'?
8A10          bne.s    FE8A5C         ; nein: Graphik ausgeben ->
8A12          movea.l a4,a0           ; a0 -> Blockanfang
8A14          move.w   #FF,d1         ; d1 := 255 = Zahl der Langworte - 1
8A18          moveq    #0,d0          ; d0 löschen
8A1A FE8A1A   add.l    (a0)+,d0       ; nächstes Langwort im Block addieren
8A1C          bcc.s    FE8A20         ; kein Übertrag: ->
```

```
8A1E         addq.l  #1,dØ            ; Übertrag addieren
8A2Ø FE8A2Ø  dbra    d1,FE8A1A        ; bis alle Langworte addiert: ->
8A24         not.l   dØ               ; Ergebnis negieren
8A26         bne.s   FE8A5C           ; nicht Ø: Graphik ausgeben ->
8A28         lea     2C(a5),a1        ; a1 -> IORequest
8A2C         jsr     C(a4)            ; ---> DOS suchen, aØ := rt_init
8A3Ø         tst.l   dØ               ; gefunden?
8A32         beq.s   FE8A56           ; ja: ->
8A34         move.l  dØ,-(a7)
8A36         movea.l a7,a1
8A38         movem.l d7/a5-a6,-(a7)   ; Register retten
8A3C         move.l  #3ØØØØØØ1,d7     ; d7 := Alert-Kode
8A42         lea     (a1),a5
8A44         movea.l 4,a6             ; a6 := SysBase
8A48         jsr     -6C(a6)          ; ---> Alert
8A4C         movem.l (a7)+,d7/a5-a6   ; Register wiederherstellen
8A5Ø         addq.l  #4,a7
8A52         bra     FE8BØØ           ; --->
8A56
8A56 FE8A56  movea.l aØ,a3            ; a3 := aØ = DOS-Init-Routine
8A58         bra     FE8BØØ           ; ---> Abschluß
8A5C
8A5C FE8A5C  move.l  4(a5),dØ         ; dØ := GfxBase oder Ø
8A6Ø         bne.s   FE8A66           ; Graphik schon ausgegeben: ->
8A62         bsr     FE8B7E           ; ---> Graphik ausgeben
8A66 FE8A66  move.w  #81ØØ,DMACON     ; Bit Plane DMA freigeben
8A6E         lea     2C(a5),a1        ; a1 -> IORequest
8A72         move.w  #9,1C(a1)        ; io_Command := td_Motor
8A78         clr.l   24(a1)           ; io_Length := Ø
8A7C         jsr     -1C8(a6)         ; ---> DoIO
8A8Ø         tst.l   dØ               ; Fehler?
8A82         bne.s   FE8AC8           ; ja: ->
8A84 FE8A84  lea     2C(a5),a1        ; a1 -> IORequest
8A88         move.w  #D,1C(a1)        ; io_Command := td_Changenum
8A8E         jsr     -1C8(a6)         ; ---> DoIO
8A92         tst.l   dØ               ; Diskette gewechselt?
8A94         bne.s   FE8AC8           ; ja: ->
8A96         cmp.l   4C(a5),d2        ; io_Actual = d2?
8A9A         beq.s   FE8A84           ; ja: ->
8A9C FE8A9C  lea     2C(a5),a1        ; a1 -> IORequest
```

```
8AA0          move.w  #E,1C(a1)        ; io_Command := td_Changestate
8AA6          jsr     -1C8(a6)         ; ---> DoIO
8AAA          tst.l   d0               ; Diskette im Laufwerk?
8AAC          bne.s   FE8AC8           ; nein: ->
8AAE          tst.l   4C(a5)           ; io_Actual = 0?
8AB2          bne.s   FE8A9C           ; nein: ->
8AB4          bra     FE89B0           ; --->
8AB8
8AB8 FE8AB8   lea     2C(a5),a1        ; a1 -> IORequest
8ABC          move.w  #D,1C(a1)        ; io_Command := td_Changenum
8AC2          jsr     -1C8(a6)         ; ---> DoIO
8AC6          bra.s   FE8A5C           ; --->
8AC8
8AC8 FE8AC8   cmpi.b  #1D,4B(a5)       ; io_Error = tderr_DiskChanged?
8ACE          beq.s   FE8AB8           ; ja: ->
8AD0          pea     0
8AD4          move.w  48(a5),2(a7)     ; io_Command auf Stack
8ADA          pea     0
8ADE          move.b  4B(a5),3(a7)     ; io_Error auf Stack
8AE4          movea.l a7,a1
8AE6          movem.l d7/a5-a6,-(a7)   ; Register retten
8AEA          move.l  #30068014,d7     ; d7 := Alert-Kode
8AF0          lea     (a1),a5
8AF2          movea.l 4,a6             ; a6 := SysBase
8AF6          jsr     -6C(a6)          ; ---> Alert
8AFA          movem.l (a7)+,d7/a5-a6   ; Register wiederherstellen
8AFE          addq.l  #8,a7            ; Parameter vom Stack nehmen
8B00 FE8B00   bsr     FE8DD0           ; ---> Graphik löschen
8B04          move.w  #8100,DMACON     ; Bit Plane DMA freigeben
8B0C          lea     2C(a5),a1        ; a1 -> IORequest
8B10          jsr     -1C2(a6)         ; ---> CloseDevice
8B14 FE8B14   moveq   #0,d0            ; d0 := 0
8B16          move.b  F(a5),d0
8B1A          jsr     -150(a6)         ; ---> FreeSignal
8B1E FE8B1E   movea.l a4,a1            ; a1 -> Puffer
8B20 FE8B20   move.l  #488,d0          ; d0 := Pufferlänge 1160
8B26          jsr     -D2(a6)          ; ---> FreeMem
8B2A FE8B2A   adda.l  #7E,a5           ; Stackbereich freimachen
8B30          unlk    a5
8B32          movea.l a3,a0            ; a0 := a3 -> DOS-Init-Routine
```

```
8B34         movem.l (a7)+,d2-d3/a3-a5  ; Register wiederherstellen
8B38         jmp     (aØ)               ; ---> DOS initialisieren
8B3A
8B3A FE8B3A  rts
8B3C
8B3C;------  Graphik-Routinen
8B3C
8B3C FE8B3C  DC.B    'graphics.library',Ø,Ø
8B4E
8B4E FE8B4E  DC.W    ØFFF,ØFFF,ØFFF,ØFFF,ØFFF,ØFFF,ØFFF,ØFFF  ; Color-Tabellen
8B5E         DC.W    ØFFF,ØFFF,ØFFF,ØFFF,ØFFF,ØFFF,ØFFF,ØFFF
8B6E         DC.W    ØFFF,ØFFF,ØFFF,ØFFF
8B76
8B76 FE8B76  DC.W    ØFFF,Ø,Ø77C,ØBBB
8B7E
8B7E ;------ Graphik ausgeben
8B7E
8B7E FE8B7E  lea     -44(pc),a1         ; a1 := FE8B3C -> 'graphics.library'
8B82         moveq   #Ø,dØ              ; Version
8B84         jsr     -228(a6)           ; ---> OpenLibrary
8B88         move.l  dØ,4(a5)           ; LibBase eintragen
8B8C         bne.s   FE8BA6             ; Library gefunden: ->
8B8E         movem.l d7/a5-a6,-(a7)     ; Register retten
8B92         move.l  #3ØØ38ØØ2,d7       ; d7 := Alert-Kode
8B98         movea.l 4,a6               ; a6 := SysBase
8B9C         jsr     -6C(a6)            ; ---> Alert
8BAØ         movem.l (a7)+,d7/a5-a6     ; Register wiederherstellen
8BA4         rts
8BA6
8BA6 FE8BA6  movem.l d2-d5/a2-a3/a6,-(a7)  ; Register retten
8BAA         movea.l Ø(a5),a6           ; a6 := SysBase
8BAE         move.l  #5E9A,dØ           ; dØ := 24218 = Bedarf an Bytes
8BB4         move.l  #1ØØØ3,d1          ; d1 := Speichertyp PUBLIC, CHIP, CLEAR
8BBA         jsr     -C6(a6)            ; ---> AllocMem
8BBE         tst.l   dØ                 ; Speicher reserviert?
8BCØ         bne     FE8BEØ             ; ja: ->
8BC4         movem.l d7/a5-a6,-(a7)     ; Register retten
8BC8         move.l  #3ØØ1ØØØØ,d7       ; d7 := Alert-Kode
8BCE         movea.l 4,a6               ; a6 := SysBase
8BD2         jsr     -6C(a6)            ; ---> Alert
```

```
8BD6        movem.l (a7)+,d7/a5-a6     ; Register wiederherstellen
8BDA        movem.l (a7)+,d2-d5/a2-a3/a6
8BDE        rts
8BE0
8BE0 FE8BE0 move.l  d0,8(a5)           ; 0    -> ViewPort
8BE4        addi.l  #28,d0
8BEA        move.l  d0,C(a5)           ; 40   -> View Structure
8BEE        addi.l  #12,d0
8BF4        move.l  d0,10(a5)          ; 58   -> RastPort Structure
8BF8        addi.l  #64,d0
8BFE        move.l  d0,14(a5)          ; 158  -> TmpRas
8C02        addi.l  #8,d0
8C08        move.l  d0,18(a5)          ; 166  -> RasInfo
8C0C        addi.l  #C,d0
8C12        move.l  d0,1C(a5)          ; 178  -> BitMap
8C16        addi.l  #28,d0
8C1C        move.l  d0,20(a5)          ; 206  -> BM_Plane0
8C20        move.l  #1F40,d1
8C26        add.l   d1,d0
8C28        move.l  d0,24(a5)          ; 8206 -> BM_Plane1
8C2C        add.l   d1,d0
8C2E        move.l  d0,28(a5)          ; 16206
8C32        movea.l 4(a5),a6           ; a6 := GfxBase
8C36        movea.l 8(a5),a0           ; a0 -> ViewPort
8C3A        jsr     -CC(a6)            ; ---> InitVPort(a0)
8C3E        movea.l C(a5),a1           ; a1 -> View
8C42        jsr     -168(a6)           ; ---> InitView(a1)
8C46        movea.l 1C(a5),a0          ; a0 -> BitMap
8C4A        moveq   #2,d0              ; d0 := Depth  = 2
8C4C        move.l  #140,d1            ; d1 := Width  = 320 px
8C52        move.l  #C8,d2             ; d2 := Height = 200 px
8C58        jsr     -186(a6)           ; ---> InitBitMap
8C5C        movea.l 1C(a5),a0          ; a0 -> BitMap
8C60        move.l  20(a5),8(a0)       ; BM_Planes eintragen
8C66        move.l  24(a5),C(a0)
8C6C        movea.l 10(a5),a1          ; a1 -> RastPort
8C70        jsr     -C6(a6)            ; ---> InitRastPort
8C74        movea.l 14(a5),a0          ; a0 -> TmpRas
8C78        movea.l 28(a5),a1          ; a1 -> Buff
8C7C        move.l  #1F40,d0           ; d0 := Size = 8000
```

```
8C82          jsr      -1D4(a6)        ; ---> InitTmpRas
8C86          movea.l  18(a5),a0       ; a0 -> RasInfo
8C8A          move.l   1C(a5),4(a0)    ; Zgr auf BitMap eintragen
8C90          movea.l  10(a5),a0       ; a0 -> RastPort
8C94          move.l   1C(a5),4(a0)    ; Zgr auf BitMap eintragen
8C9A          move.l   14(a5),C(a0)    ; Zgr auf TmpRas eintragen
8CA0          movea.l  8(a5),a0        ; a0 -> ViewPort
8CA4          move.w   #C8,1A(a0)      ; vp_DHeight := 200
8CAA          move.w   #140,18(a0)     ; vp_DWidth := 320
8CB0          move.l   18(a5),24(a0)   ; vp_RasInfo eintragen
8CB6          clr.w    20(a0)          ; vp_Modes := 0
8CBA          movea.l  C(a5),a3        ; a3 -> View
8CBE          move.l   8(a5),0(a3)     ; ViewPort eintragen
8CC4          movea.l  a3,a0           ; a0 -> View
8CC6          movea.l  8(a5),a1        ; a1 -> ViewPort
8CCA          jsr      -D8(a6)         ; ---> MakeVPort
8CCE          movea.l  a3,a1           ; a1 -> View
8CD0          jsr      -D2(a6)         ; ---> MrgCop
8CD4          movea.l  a3,a1           ; a1 -> View
8CD6          jsr      -DE(a6)         ; ---> LoadView
8CDA          movea.l  8(a5),a0        ; a0 -> ViewPort
8CDE          lea      -192(pc),a1     ; a1 := FE8B4E -> Colors
8CE2          moveq    #14,d0          ; d0 := 20 Worte
8CE4          jsr      -C0(a6)         ; ---> LoadRGB4
8CE8          movea.l  10(a5),a3       ; a3 -> RastPort
8CEC          lea      12E(pc),a2      ; a2 := FE8E1C
8CF0          movea.l  a3,a1           ; a1 -> RastPort
8CF2          moveq    #0,d0           ; d0 := 0 = Mode
8CF4          jsr      -162(a6)        ; ---> SetDrMd
8CF8 FE8CF8   moveq    #0,d3           ; d3 := 0
8CFA          move.b   (a2)+,d3        ; d3 := nächstes Byte aus Tabelle
8CFC          moveq    #0,d5           ; d5 := 0
8CFE          move.b   (a2)+,d5        ; d5 := nächstes Byte aus Tabelle
8D00          cmpi.b   #FF,d3          ; d3 = $FF?
8D04          bne.s    FE8D2E          ; nein: ->
8D06          cmpi.b   #FF,d5          ; d5 = $FF?
8D0A          beq      FE8D6A          ; ja: ->
8D0E          moveq    #0,d4           ; d4 := 0
8D10          move.b   (a2)+,d4        ; d4 := nächstes Byte aus Tabelle
8D12          moveq    #0,d3           ; d3 := 0
```

```
8D14        move.b   (a2)+,d3        ; d3 := nächstes Byte aus Tabelle
8D16        movea.l  a3,a1           ; a1 -> RastPort
8D18        move.l   d5,d0           ; d0 := d5 = color aus Tabelle
8D1A        jsr      -156(a6)        ; ---> SetBPen
8D1E        moveq    #28,d1          ; d1 := 40
8D20        add.l    d3,d1           ; d1 := d3+40 = y
8D22        moveq    #46,d0          ; d0 := 70
8D24        add.l    d4,d0           ; d0 := d4+70 = x
8D26        movea.l  a3,a1           ; a1 -> RastPort
8D28        jsr      -F0(a6)         ; ---> Move
8D2C        bra.s    FE8CF8          ; ---> Loop
8D2E
8D2E FE8D2E cmpi.b   #FE,d3          ; d3 = $FE
8D32        bne.s    FE8D56          ; nein: ->
8D34        moveq    #0,d4           ; d4 := 0
8D36        move.b   (a2)+,d4        ; d4 := nächstes Byte aus Tabelle
8D38        moveq    #0,d3           ; d3 := 0
8D3A        move.b   (a2)+,d3        ; d3 := nächstes Byte aus Tabelle
8D3C        movea.l  a3,a1           ; a1 -> RastPort
8D3E        move.l   d5,d0           ; d0 := d5 = color aus Tabelle
8D40        jsr      -156(a6)        ; ---> SetAPen
8D44        moveq    #28,d1          ; d1 := 40
8D46        add.l    d3,d1           ; d1 := d3+40 = y
8D48        moveq    #46,d0          ; d0 := 70
8D4A        add.l    d4,d0           ; d0 := d4+70 = x
8D4C        moveq    #1,d2           ; d2 := Mode = 1
8D4E        movea.l  a3,a1           ; a1 -> RastPort
8D50        jsr      -14A(a6)        ; ---> Flood
8D54        bra.s    FE8CF8          ; ---> Loop
8D56
8D56 FE8D56 move.l   d3,d4           ; d4 := d3
8D58        move.l   d5,d3           ; d3 := d5
8D5A        moveq    #28,d1          ; d1 := 40
8D5C        add.l    d3,d1           ; d1 := d3+40 = y
8D5E        moveq    #46,d0          ; d0 := 70
8D60        add.l    d4,d0           ; d0 := d4+70 = x
8D62        movea.l  a3,a1           ; a1 -> RastPort
8D64        jsr      -F6(a6)         ; ---> Draw
8D68        bra.s    FE8CF8          ; ---> Loop
8D6A
```

```
8D6A FE8D6A  lea      24C(pc),a2        ; a2 := FE8B2Ø
8D6E         movea.l  a3,a1             ; a1 -> RastPort
8D7Ø         moveq    #3,dØ             ; dØ := color
8D72         jsr      -156(a6)          ; ---> SetAPen
8D76 FE8D76  move.w   (a2)+,dØ          ; dØ := nächstes Wort aus Tabelle
8D78         bmi.s    FE8DB8            ; Bit 15 gesetzt: ->
8D7A         move.b   dØ,18(a3)         ; rp_Mask in RastPort eintragen
8D7E         moveq    #Ø,d4             ; d4 := Ø
8D8Ø         move.b   (a2)+,d4          ; d4 := nächstes Byte aus Tabelle
8D82         moveq    #Ø,d5             ; d5 := Ø
8D84         move.b   (a2)+,d5          ; d5 := nächstes Byte aus Tabelle
8D86         moveq    #46,d2            ; d2 := 7Ø
8D88         moveq    #Ø,dØ             ; dØ := Ø
8D8A         move.b   (a2)+,dØ          ; dØ := nächstes Byte aus Tabelle
8D8C         add.l    dØ,d2             ; d2 := dØ+7Ø = x
8D8E         moveq    #28,d3            ; d3 := 4Ø
8D9Ø         move.b   (a2)+,dØ          ; dØ := nächstes Byte aus Tabelle
8D92         add.l    dØ,d3             ; d3 := dØ+4Ø = y
8D94         move.w   d4,dØ             ; dØ := d4
8D96         mulu     d5,dØ             ; dØ := dØ mal d5 = Zähler
8D98         lea      4Ø8(a4),aØ        ; aØ -> Platz für Schablone
8D9C         bra.s    FE8DAØ            ; --->
8D9E
8D9E FE8D9E  move.w   (a2)+,(aØ)+       ; Schablone aus Tabelle erzeugen
8DAØ FE8DAØ  dbra     dØ,FE8D9E
8DA4         lea      4Ø8(a4),aØ        ; aØ -> Schablone
8DA8         moveq    #Ø,dØ             ; dØ := SrcX = Ø
8DAA         add.l    d4,d4             ; d4 verdoppeln
8DAC         move.l   d4,d1             ; d1 := SrcMod
8DAE         movea.l  a3,a1             ; a1 -> RastPort
8DBØ         lsl.w    #3,d4             ; d4 := SizeX
8DB2         jsr      -24(a6)           ; ---> BltTemplate
8DB6         bra.s    FE8D76            ; --->
8DB8
8DB8 FE8DB8  movea.l  8(a5),aØ          ; aØ -> ViewPort
8DBC         lea      -248(pc),a1       ; a1 := FE8B76 = Color-Tabelle
8DCØ         moveq    #4,dØ             ; dØ := 4 = Zahl der Worte
8DC2         jsr      -CØ(a6)           ; ---> LoadRGB4
8DC6         jsr      -1ØE(a6)          ; ---> WaitTOF
8DCA         movem.l  (a7)+,d2-d5/a2-a3/a6
```

```
8DCE        rts
8DD0
8DD0 ;------ Graphik löschen
8DD0
8DD0 FE8DD0  move.l  a6,-(a7)          ; a6 retten
8DD2         tst.l   4(a5)             ; GfxBase gespeichert?
8DD6         beq.s   FE8E16            ; Nein: fertig ->
8DD8         tst.l   8(a5)             ; ViewPort-Zeiger vorhanden?
8DDC         beq.s   FE8E0A            ; Nein: fertig ->
8DDE         move.w  #100,DMACON       ; BitPlane DMA sperren
8DE6         movea.l 4(a5),a6          ; a6 := GfxBase
8DEA         suba.l  a1,a1             ; a1 := 0
8DEC         jsr     -DE(a6)           ; ---> LoadView
8DF0         movea.l 8(a5),a0          ; a0 -> ViewPort
8DF4         jsr     -21C(a6)          ; ---> FreeVPortCopLists
8DF8         movea.l 0(a5),a6          ; a6 := SysBase
8DFC         movea.l 8(a5),a1          ; a1 -> ViewPort Memory
8E00         move.l  #5E9A,d0          ; d0 := 24218 = Anzahl Bytes
8E06         jsr     -D2(a6)           ; ---> FreeMem
8E0A FE8E0A  movea.l 0(a5),a6          ; a6 := SysBase
8E0E         movea.l 4(a5),a1          ; a1 := GfxBase
8E12         jsr     -19E(a6)          ; ---> CloseLibrary
8E16 FE8E16  movea.l (a7)+,a6          ; a6 wiederherstellen
8E18         rts
8E1A
8E1A         DC.W    0
8E1C ;------  Graphik-Tabelle für Hand mit Workbench-Disk
8E1C
8E1C ; Alle Koordinatenwerte beziehen sich auf x0 = 40, y0 = 70
8E1C
8E1C FE8E1C  DC.B    FF,01,23,0B         ; Move, Color, x, y
8E20         DC.B    3A,0B,3A,21,71,21,71,0B,7D,0B,88,16,88,5E,7F,5E  ; Draw
8E30         DC.B    7F,38,40,38,3E,36,35,36,34,38,2D,38,2D,41,23,48
8E40         DC.B    23,0B
8E42         DC.B    FE,02,25,45         ; Flood, Color, x, y
8E46         DC.B    FF,01,21,48         ; Move, Color, x, y
8E4A         DC.B    21,0A,7E,0A,8A,16,8A,5F,56,5F,56,64,52,6C,4E,71  ; Draw
8E5A         DC.B    4A,74,44,7D,3C,81,3C,8C,0A,8C,0A,6D,09,6D,09,51
8E6A         DC.B    0D,4B,14,45,15,41,19,3A,1E,37,21,36,21,36,1E,38
8E7A         DC.B    1A,3A,16,41,15,45,0E,4B,0A,51,0A,6C,0B,6D,0B,8B
```

```
8E8A        DC.B    28,8B,28,76,3Ø,76,34,72,34,5F,32,5C,32,52,41,45
8E9A        DC.B    41,39,3E,37,3B,37,3E,3A,3E,41,3D,42,36,42,33,3F
8EAA        DC.B    2A,46,1E,4C,12,55,12,54,1E,4B,1A,4A,17,47,2A,46
8EBA        DC.B    1E,4A,21,48
8EBE        DC.B    FF,Ø1,32,3D          ; Move, Color, x, y
8EC2        DC.B    34,36,3C,37,3D,3A,3D,41,36,41,32,3D  ; Draw
8ECE        DC.B    FF,Ø1,33,5C          ; Move, Color, x, y
8ED2        DC.B    33,52,42,45,42,39,7D,39,7D,5E,34,5E,33,5A  ; Draw
8EEØ        DC.B    FF,Ø1,3C,ØB          ; Move, Color, x, y
8EE4        DC.B    6F,ØB,6F,2Ø,3C,2Ø,3C,ØB  ; Draw
8EEC        DC.B    FF,Ø1,6Ø,ØE          ; Move, Color, x, y
8EFØ        DC.B    6B,ØE,6B,1C,6Ø,1C,6Ø,ØE  ; Draw
8EF8        DC.B    FE,Ø3,3E,1F          ; Flood, Color, x, y
8EFC        DC.B    FF,Ø1,62,ØF          ; Move, Color, x, y
8FØØ        DC.B    69,ØF,69,1B,62,1B,62,ØF  ; Draw
8FØ8        DC.B    FE,Ø2,63,1A          ; Flood, Color, x, y
8FØC        DC.B    FF,Ø1,2F,39          ; Move, Color, x, y
8F1Ø        DC.B    32,39,32,3B,2F,3F,2F,39  ; Draw
8F18        DC.B    FF,Ø1,29,8B          ; Move, Color, x, y
8F1C        DC.B    29,77,3Ø,77,35,72,35,69,39,6B,41,6B,41,6D,45,72  ; Draw
8F2C        DC.B    49,72,49,74,43,7D,3B,8Ø,3B,8B,29,8B
8F38        DC.B    FF,Ø1,35,5F          ; Move, Color, x, y
8F3C        DC.B    35,64,3A,61,35,5F    ; Draw
8F42        DC.B    FF,Ø1,39,62          ; Move, Color, x, y
8F46        DC.B    35,64,35,5F,4A,5F,4Ø,69,3F,69,41,67,3C,62,39,62  ; Draw
8F56        DC.B    FF,Ø1,4E,5F          ; Move, Color, x, y
8F5A        DC.B    55,5F,55,64,51,6C,4E,7Ø,49,71,46,71,43,6D,43,6A  ; Draw
8F6A        DC.B    4E,5F
8F6C        DC.B    FF,Ø1,44,6A          ; Move, Color, x, y
8F7Ø        DC.B    44,6D,46,7Ø,48,7Ø,4C,6F,4D,6C,49,69,44,6A  : Draw
8F7E        DC.B    FF,Ø1,36,68          ; Move, Color, x, y
8F82        DC.B    3E,6A,4Ø,67,3C,63,39,63,36,65,36,68  ; Draw
8F8E        DC.B    FF,Ø1,7E,ØB          ; Move, Color, x, y
8F92        DC.B    89,16,89,5E          ; Draw
8F96        DC.B    FE,Ø1,22,ØB          ; Flood, Color, x, y
8F9A        DC.B    FE,Ø1,3B,ØB          ; Flood, Color, x, y
8F9E        DC.B    FE,Ø1,61,ØF          ; Flood, Color, x, y
8FA2        DC.B    FE,Ø1,6A,1B          ; Flood, Color, x, y
8FA6        DC.B    FE,Ø1,7Ø,ØF          ; Flood, Color, x, y
8FAA        DC.B    FE,Ø1,7E,5E          ; Flood, Color, x, y
```

```
8FAE        DC.B   FE,01,4B,60          ; Flood, Color, x, y
8FB2        DC.B   FE,01,2E,39          ; Flood, Color, x, y
8FB6        DC.B   FF,FF                ; Anfangsmarke für Muster
8FB8 FE8FB8 DC.W   2                    ; rp_Mask
8FBA        DC.B   04,08,39,54          ; SizeX, SizeY, DestX, DestY
8FBE        DC.W   1001,8220,4050,0004  ; Muster 2 'Amiga'
8FC6        DC.W   0404,8889,1210,5010
8FCE        DC.W   0408,2448,1090,1020
8FD6        DC.W   0410,1042,0510,1040
8FDE        DC.W   0420,0050,2690,1080
8FE6        DC.W   0440,0484,0900,1100
8FEE        DC.W   0480,4960,4208,1200
8FF6        DC.W   0500,0683,8404,1400
8FFE        DC.W   1                    ; rp_Mask
9000        DC.B   04,08,39,54          ; SizeX, SizeY, DestX, DestY
9004        DC.W   1F9F,FE3F,FF77,FE7C  ; Muster 1 'Amiga'
900C        DC.W   070C,EF8F,1E71,DC30
9014        DC.W   07F8,E7CE,1CF3,1FE0
901C        DC.W   0731,F3CE,1DF6,1CC0
9024        DC.W   0760,03DC,3FFC,1D80
902C        DC.W   07C0,879C,3F78,1F00
9034        DC.W   0780,CF78,7E78,1E00
903C        DC.W   0700,FEFF,FC7C,1C00
9044        DC.W   2                    ; rp_Mask
9046        DC.B   03,07,4A,4A          ; SizeX, SizeY, DestX, DestY
904A        DC.W   C707,8F8C,3000       ; Muster 'Work'
9050        DC.W   6603,18CE,7000
9056        DC.W   3E63,18CF,F000
905C        DC.W   6667,18CD,B000
9062        DC.W   C63B,8F8C,3000
9068        DC.W   0600,000C,3000
906E        DC.W   0700,000C,3000
9074        DC.W   2                    ; rp_Mask
9076        DC.B   03,07,46,40          ; SizeX, SizeY, DestX, DestY
907A        DC.W   C71F,18C7,C3E0       ; Muster 'bench'
9080        DC.W   C631,98C0,6630
9086        DC.W   C601,98CF,E630
908C        DC.W   CE31,98CC,6670
9092        DC.W   761F,0FC7,C3B0
9098        DC.W   0600,0000,0030
```

```
909E        DC.W    0700,0000,0038
90A4        DC.W    1                   ; rp_Mask
90A6        DC.B    03,0A,5C,64         ; SizeX, SizeY, DestX, DestY
90AA        DC.W    0F9F,C1C0,0FC0      ; Muster 'V 1.2'
90B0        DC.W    0707,03C0,1FE0
90B6        DC.W    070E,07C0,3CF0
90BC        DC.W    071C,01C0,0070
90C2        DC.W    0738,01C0,00F0
90C8        DC.W    0770,01C0,03E0
90CE        DC.W    07E0,01C0,0780
90D4        DC.W    07C0,01C0,0F00
90DA        DC.W    0780,07F3,3FF0
90E0        DC.W    0700,07F3,3FF0
90E6        DC.W    FFFF                ; Endemarke
90E8
90E8 ************************ Ende von strap *********************************
```

DOS - Bootblock

```
0000 *********************** DOS-Bootblock ********************************
0000
0000 DOS000  DC.B     'DOS',0            ; Kennmarke für DOS-Bootblock
0004
0004         DC.L     B2CAD60E           ; bb_CheckSum
0008         DC.L     370                ; bb_DosBlock (reserviert)
000C
000C DOS00C  lea      18(pc),a1          ; a1 := DOS026 -> 'dos.library'
0010         jsr      -60(a6)            ; ---> FindResident
0014         tst.l    d0                 ; gefunden?
0016         beq.s    DOS022             ; nein: Fehler ->
0018         movea.l  d0,a0              ; a0 := d0 -> DOS-Modul
001A         movea.l  16(a0),a0          ; a0 -> DOS-Initialisierungsroutine
001E         moveq    #0,d0              ; d0 := 0 = ok-Flag
0020 DOS020  rts
0022
0022 DOS022  moveq    #-1,d0             ; d0 := -1 = Fehlerflag
0024         bra.s    DOS020             ; ---> rts
0026
0026 DOS026  DC.B     'dos.library',0
0032
0032 *******************************************************************
```

Amiga . Bootrom

```
0000 *********************************************************************
0000 *                                                                   *
0000 *                A M I G A . B O O T R O M                          *
0000 *                                                                   *
0000 *********************************************************************
0000
0000 F80000  DC.W    1111
0002
0002         jmp     F8008A             ; --->
0008
0008         DC.L    0
000C F8001C  DC.L    -1
0010         DC.L    -1
0014         DC.L    -1
0018         DC.W    19
001A         DC.W    9
001C
001C         DC.B    0D,0A,0A,'Commodore AMIGA ROM Bootstrap',0D,0A
003E         DC.B    'Copyright (C) 1985, Commodore Amiga, Inc.',0D,0A
0069         DC.B    'All rights reserved.',0D,0A,0
0080
0080 F80080  reset
0082         movea.l F80004,a0          ; a0 := F8008A
0088         jmp     (a0)               ; --->
008A
008A F8008A  moveq   #0,d0              ; Zeitzähler := 0
008C         move.l  #A,d1              ; Blinkzähler := 10
0092         move.b  #2,BFE201          ; CIAA DDRA: LED-Ausgang setzen
009A         move.b  #2,BFE001          ; CIAA PRA:  LED dunkel
00A2 F800A2  dbra    d0,F800A2          ; Warteschleife ca. 0.1 s
00A6         bchg.b  #1,BFE001          ; LED Hell-Dunkel-Wechsel
```

```
00AE          dbra     d1,F800A2          ; 10 mal wiederholen ->
00B2          reset
00B4          move.w   #444,DFF180        ; COLOR00: Bildschirm dunkelgrau
00BC          lea      40000,a7           ; Stack-Zeiger initialisieren
00C2          lea      -C4(pc),a0         ; a0 := F80000
00C6          cmpa.l   #F00000,a0         ; Anfang bei $F00000?
00CC          beq.s    F000EE             ; Ja: ->
00CE          cmpi.w   #1111,F00000       ; ROM-Kennzeichen bei $F00000?
00D6          bne.s    F800EE             ; Nein: ->
00D8          move.w   #FFF,DFF180        ; COLOR00: Bildschirm weiß
00E0          lea      F800EE,a5          ; a5 := Rückkehradresse
00E6          movea.l  F00004,a0          ; a0 := Zieladresse
00EC          jmp      (a0)               ; --->
00EE
00EE F800EE   move.b   #3,BFE201          ; CIAA DDRA: LED- u. OVL = Ausgang
00F6          move.b   #2,BFE001          ; CIAA PRA:  LED dunkel
00FE          movea.l  #DFF000,a4         ; a4 := Spezialchip-Basisadresse
0104          move.w   #7FFF,d6           ; d6 := Löschmaske
0108          move.w   d6,9A(a4)          ; INTENA: Alle Bits löschen
010C          move.w   d6,9C(a4)          ; INTREQ: Alle Bits löschen
0110          move.w   d6,96(a4)          ; DMACON: Alle Bits löschen
0114          lea      -FA(pc),a0         ; a0 := F8001C -> IDString
0118          lea      F80122,a5          ; a5 := Rückkehradresse
011E          bra      F80284             ; ---> jmp (a5)
0122
0122 F80122   lea      FC0000,a0          ; a0 := Kickstart-ROM-Anfang
0128          cmpi.l   #11114EF9,(a0)     ; steht dort '1111' und 'jmp'?
012E          bne.s    F80144             ; Nein: Kickstart nicht geladen ->
0130          move.l   #10000,d1          ; Langwort-Zähler initialisieren
0136          lea      6(pc),a5           ; a5 := F8013E = Rückkehradresse
013A          bra      F802A8             ; ---> Prüfsumme berechnen
013E
013E F8013E   tst.l    d0                 ; Prüfsumme o.k.?
0140          beq      F80262             ; Ja: Exec starten ->
0144 F80144   lea      -146(pc),a0        ; a0 := ROM-Anfangsadresse
0148          cmpa.l   #F80000,a0         ; ROM-Anfang bei F80000?
014E          bne.s    F80196             ; Nein: ->
0150          lea      FC0000,a2          ; a2 := Anfang für RAM-Test
0156          lea      FFFFFE,a3          ; a3 := Ende für RAM-Test
015C          lea      6(pc),a5           ; a5 := F80164 = Rückkehradresse
```

```
0160        bra     F802C6           ; ---> RAM-Test Kickstart-Bereich
0164
0164 F80164 tst.l   d0               ; RAM-Test o.k.?
0166        beq.s   F80196           ; Ja: ->
0168        move.w  #CC,DFF180       ; COLOR00: Bildschirm blaugrün
0170        lea     C(pc),a0         ; a0 := F8017E -> Fehlertext
0174        lea     F80192,a5        ; a5 := Zieladresse
017A        bra     F80284           ; ---> jmp (a5)
017E
017E F8017E DC.B    0D,0A,'ram-rom-failure',0D,0A,0
0192
0192 F80192 bra     F80080           ; ---> Neustart
0196
0196 F80196 lea     0,a2             ; a2 := Anfang für RAM-Test
019A        lea     3FFFE,a3         ; a3 := Ende für RAM-Test
01A0        lea     6(pc),a5         ; a5 := F801A8 = Rückkehradresse
01A4        bra     F80360           ; ---> RAM-Test für Chip Memory
01A8
01A8 F801A8 tst.l   d0               ; RAM-Test o.k.?
01AA        beq.s   F801DC           ; Ja: ->
01AC        move.w  #C0,DFF180       ; COLOR00: Bildschirm grün
01B4        lea     C(pc),a0         ; a0 := F801C2 -> Fehlertext
01B8        lea     F801D8,a5        ; a5 := Zieladresse
01BE        bra     F80284           ; ---> jmp (a5)
01C2
01C2 F801C2 DC.B    0D,0A,'local-ram-failure',0D,0A,0
01D8
01D8 F801D8 bra     F80080           ; ---> Neustart
01DC
01DC F801DC lea     6(pc),a5         ; a5 := F801E4 = Rückkehradresse
01E0        bra     F807A2           ; ---> Audio-Test mit Melodie
01E4
01E4 F801E4 movea.w #8,a0            ; a0 -> Anfang der Ausnahmevektoren
01E8        move.w  #2D,d1           ; d1 := Zahl der Vektoren
01EC        lea     A(pc),a1         ; a1 := F801F8 -> Ausnahmeroutine
01F0 F801F0 move.l  a1,(a0)+         ; Ausnahmevektoren #2 bis #$2F
01F2        dbra    d1,F801F0        ; einrichten
01F6        bra.s   F80226           ; --->
01F8
01F8 F801F8 move.w  #CC0,DFF180      ; COLOR00: Bildschirm gelb
```

```
0200         lea     C(pc),a0           ; a0 := F8020E -> Fehlertext
0204         lea     F80222,a5          ; a5 := Zieladresse
020A         bra     F80284             ; ---> jmp (a5)
020E
020E F8020E  DC.B    0D,0A,'boot-exception',0D,0A,0,0
0222
0222 F80222  bra     F80080             ; ---> Neustart
0226
0226 F80226  move.w  #0,DFF180          ; COLOR00: Bildschirm schwarz
022E         jsr     F80A18             ; ---> Disk-Boot
0234         tst.l   d0                 ; o.k.?
0236         beq.s   F80262             ; Ja: ->
0238         lea     C(pc),a0           ; a0 := F80246 -> Fehlertext
023C         lea     F8025E,a5          ; a5 := Zieladresse
0242         bra     F80284             ; ---> jmp (a5)
0246
0246 F80246  DC.B    0D,0A,'disk-boot-failed!?',0D,0A,0,0
025E
025E F8025E  bra     F80080             ; ---> Neustart
0262
0262 F80262  move.w  #0,DFF180          ; COLOR00: Bildschirm schwarz
026A         lea     F80000,a1          ; a1 := Boot-ROM-Anfangsadresse
0270         movea.l FC0004,a0          ; a0 := Exec-Startadresse
0276         moveq   #0,d0              ; d0 := 0
0278         move.l  F80280,-(a7)       ; Befehlszeilen auf Stack legen
027E         jmp     (a7)               ; ---> und aufrufen
0280
0280 F80280  move.w  d0,(a1)            ; Boot-ROM abschalten
0282         jmp     (a0)               ; ---> Exec starten
0284
0284 F80284  jmp     (a5)               ; --->
0286
0286         DC.W    0
0288
0288 ;-----  Prüfsumme wortweise berechnen (nicht gerufen)
0288
0288         move.l  d1,d2              ; d1.w = Wortzähler lo
028A         swap    d2                 ; d2.w = Wortzähler hi
028C         clr.l   d0                 ; d0 als Summenregister löschen
028E         bra.s   F80298             ; --->
```

```
0290
0290 F80290  add.w   (a0)+,d0        ; Wort in Summenregister addieren
0292         bcc.s   F80298          ; kein Übertrag: ->
0294         addi.w  #1,d0           ; Übertrag addieren
0298 F80298  dbra    d1,F80290       ; wiederholen, bis Wortzähler
029C         dbra    d2,F80290       ; abgelaufen
02A0         not.w   d0              ; Ergebnis invertieren
02A2         bne.s   F802A6          ; ergibt nicht Null: ->
02A4         clr.l   d0              ; d0 := 0
02A6 F802A6  jmp     (a5)            ; ---> Rückkehr
02A8
02A8 ;-----  Prüfsumme langwortweise berechnen
02A8
02A8 F802A8  move.l  d1,d2           ; d1.w = Langwortzähler lo
02AA         swap    d2              ; d2.w = Langwortzähler hi
02AC         clr.l   d0              ; d0 als Summenregister lösche
02AE         bra.s   F802BA          ; --->
02B0
02B0 F802B0  add.l   (a0)+,d0        ; Langwort in Register addieren
02B2         bcc.s   F802BA          ; kein Übertrag: ->
02B4         addi.l  #1,d0           ; Übertrag addieren
02BA F802BA  dbra    d1,F802B0       ; wiederholen, bis
02BE         dbra    d2,F802B0       ; Langwortzähler abgelaufen
02C2         not.l   d0              ; Ergebnis invertieren
02C4         jmp     (a5)            ; ---> Rücksprung
02C6
02C6 ;-----  RAM-Test 1
02C6
02C6 F802C6  move.l  a3,d3           ; d3 := a3 = Endadresse
02C8         sub.l   a2,d3           ; - Anfangsadresse = Länge in Bytes
02CA         lsr.l   #2,d3           ; d3 := Länge in Langworten
02CC         move.l  d3,d4           ; d4 := Länge in Langworten (statisch)
02CE         clr.l   d0              ; d0 := 0
02D0         move.w  #10,d0          ; d0 := 16
02D4         addq.b  #4,d0           ; d0 := 20 = Fehlerkode
02D6         move.l  #AAAAAAAA,d1    ; d1 := Bitmuster 10101010...
02DC         movea.l a2,a0           ; a0 := a2 = Anfangsadresse
02DE F802DE  move.l  d1,(a0)         ; Testmuster speichern
02E0         move.l  (a0)+,d2        ; und wieder lesen, Adresse erhöhen
02E2         cmp.l   d1,d2           ; gelesen wie gespeichert?
```

```
02E4          bne      F803E6          ; Nein: RAM-Fehler ->
02E8          dbra     d3,F802DE       ; bis Endadresse wiederholen ->
02EC          move.w   #20,d0          ; d0 := 32
02F0          addq.b   #4,d0           ; d0 := 36 = Fehlerkode
02F2          move.l   #55555555,d1    ; d1 := Bitmuster 01010101...
02F8          movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
02FA          move.l   d4,d3           ; d3 := d4 = Länge in Langworten
02FC F802FC   move.l   d1,(a0)         ; Testmuster speichern
02FE          move.l   (a0)+,d2        ; und wieder lesen
0300          cmp.l    d1,d2           ; gelesen wie gespeichert?
0302          bne      F803E6          ; Nein: RAM-Fehler ->
0306          dbra     d3,F802FC       ; bis Endadresse wiederholen ->
030A          move.w   #40,d0          ; d0 := 64 = Fehlerkode
030E          move.l   #77777777,d1    ; d1 := Bitmuster 01110111...
0314 F80314   rol.l    #1,d1           ; Testmuster 1 Bit nach links rotieren
0316          movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
0318          move.l   d4,d3           ; d3 := d4 = Länge in Langworten
031A F8031A   move.l   d1,(a0)+        ; Testmuster in Speicher schreiben
031C          dbra     d3,F8031A       ; bis Endadresse wiederholen ->
0320          movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
0322          move.l   d4,d3           ; d3 := d4 = Länge in Langworten
0324 F80324   move.l   (a0),d2         ; Langwort aus Speicher lesen
0326          cmp.l    d1,d2           ; gelesen wie gespeichert?
0328          bne      F803E6          ; Nein: RAM-Fehler ->
032C          dbra     d3,F80324       ; bis Endadresse wiederholen ->
0330          tst.l    d1              ; Testmuster schon 4 Bits rotiert?
0332          bmi.s    F80314          ; Nein: wiederholen ->
0334          move.w   #60,d0          ; d0 := 96 = Fehlerkode
0338          move.l   #88888888,d1    ; d1 := Bitmuster 10001000...
033E F8033E   rol.l    #1,d1           ; Testmuster 1 Bit nach links rotieren
0340          movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
0342          move.l   d4,d3           ; d3 := d4 = Länge in Langworten
0344 F80344   move.l   d1,(a0)+        ; Testmuster in Speicher schreiben
0346          dbra     d3,F80344       ; bis Endadresse wiederholen ->
034A          movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
034C          move.l   d4,d3           ; d3 := d4 = Länge in Langworten
034E F8034E   move.l   (a0),d2         ; Langwort aus Speicher lesen
0350          cmp.l    d1,d2           ; gelesen wie gespeichert?
0352          bne      F803E6          ; Nein: RAM-Fehler ->
0356          dbra     d3,F8034E       ; bis Endadresse wiederholen ->
```

```
035A           tst.l    d1              ; Testmuster schon 4 Bits rotiert?
035C           bpl.s    F8033E          ; Nein: wiederholen ->
035E           bra.s    F8036A          ; ---> RAM-Test 2
0360
0360 ;-----    RAM-Test 2
0360
0360 F80360    move.l   a3,d3           ; d3 := a3 = Endadresse
0362           sub.l    a2,d3           ; - Anfangsadresse = Länge in Bytes
0364           lsr.l    #2,d3           ; d3 := Länge in Langworten
0366           move.l   d3,d4           ; d4 := Länge in Langworten (statisch)
0368           clr.l    d0              ; d0 := 0
036A F8036A    move.w   #70,d0          ; d0 := 112 = Fehlerkode
036E           movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
0370           move.l   d4,d3           ; d3 := d4 = Länge in Langworten
0372 F80372    move.l   a0,(a0)+        ; Aktuelle Adresse abspeichern
0374           dbra     d3,F80372       ; bis Endadresse wiederholen ->
0378           movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
037A           move.l   d4,d3           ; d3 := d4 = Länge in Langworten
037C F8037C    move.l   a0,d1           ; d1 := a0 = aktuelle Adresse
037E           move.l   (a0)+,d2        ; d2 := Langwort aus Speicher
0380           cmp.l    d1,d2           ; gelesen wie gespeichert?
0382           bne      F803E6          ; Nein: RAM-Fehler ->
0386           dbra     d3,F8037C       ; bis Endadresse wiederholen ->
038A           move.w   #80,d0          ; d0 := 128 = Fehlerkode
038E           movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
0390           move.l   d4,d3           ; d3 := d4 = Länge in Langworten
0392 F80392    move.l   a0,d1           ; d1 := a0 = aktuelle Adresse
0394           not.l    d1              ; d1 invertieren
0396           move.l   d1,(a0)+        ; und abspeichern
0398           dbra     d3,F80392       ; bis Endadresse wiederholen ->
039C           movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
039E           move.l   d4,d3           ; d3 := d4 = Länge in Langworten
03A0 F803A0    move.l   a0,d1           ; d1 := a0 = aktuelle Adresse
03A2           not.l    d1              ; d1 invertieren
03A4           move.l   (a0)+,d2        ; d2 := Langwort aus Speicher
03A6           cmp.l    d1,d2           ; gelesen wie gespeichert?
03A8           bne      F803E6          ; Nein: RAM-Fehler ->
03AC           dbra     d3,F803A0       ; bis Endadresse wiederholen ->
03B0           move.w   #90,d0          ; d0 := 144 = Fehlerkode
03B4           movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
```

```
03B6         clr.l    d1              ; d1 := 0
03B8         move.l   d4,d3           ; d3 := d4 = Länge in Langworten
03BA         lsl.l    #2,d3           ; d3 := Länge in Bytes
03BC         move.l   d3,d5           ; d5 := d3 = Länge in Bytes
03BE         swap     d5              ; d5.w := Länge in Bytes (Übertrag)
03C0 F803C0  move.b   d1,(a0)+        ; Speicher byteweise löschen
03C2         dbra     d3,F803C0       ; wiederholen, bis d3.w = -1 ->
03C6         dbra     d5,F803C0       ; wiederholen, bis d5.w = -1 ->
03CA         movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
03CC         move.l   d4,d3           ; d3 := d4 = Länge in Langworten
03CE         subq.l   #1,d3           ; minus 1
03D0         lsl.l    #1,d3           ; d3 := Länge in Worten
03D2         move.l   d3,d5           ; d5 := d3 = Länge in Worten
03D4         swap     d5              ; d5.w := Länge in Worten (Übertrag)
03D6 F803D6  move.w   (a0)+,d2        ; d2 := Wort aus Speicher
03D8         cmp.w    d1,d2           ; gelesen wie gespeichert?
03DA         bne.s    F803E6          ; Nein: RAM-Fehler ->
03DC         dbra     d3,F803D6       ; wiederholen, bis d3.w = -1 ->
03E0         dbra     d5,F803D6       ; wiederholen, bis d5.w = -1 ->
03E4         clr.l    d0              ; d0 := 0 = ok-Flag
03E6 F803E6  suba.w   #4,a0           ; a0 -> erste Fehlerstelle
03EA         jmp      (a5)            ; ---> Rücksprung
03EC
03EC ;-----  RAM-Test 3 (zerstörungsfrei; nicht gerufen)
03EC
03EC         movem.l  d2-d3,-(a7)     ; Register retten
03F0         move.l   a3,d3           ; d3 := a3 = Endadresse
03F2         sub.l    a2,d3           ; - Anfangsadresse = Länge in Bytes
03F4         lsr.l    #2,d3           ; d3 := Länge in Langworten
03F6         movea.l  a2,a0           ; a0 := a2 = Anfangsadresse
03F8         clr.l    d0              ; d0 := 0
03FA         move.w   #A0,d0          ; d0 := 160 = Fehlerkode
03FE F803FE  move.l   a0,d1           ; d1 := a0 = aktuelle Adresse
0400         move.l   (a0),d2         ; Speicher-Langwort in d2 retten
0402         move.l   d1,(a0)         ; d1 in Speicher schreiben
0404         cmp.l    (a0),d1         ; und mit Speicherinhalt vergleichen
0406         bne      F8041C          ; ungleich: RAM-Fehler ->
040A         not.l    d1              ; d1 invertieren
040C         move.l   d1,(a0)         ; wieder in Speicher schreiben
040E         cmp.l    (a0),d1         ; und mit Speicherinhalt vergleichen
```

```
Ø41Ø         bne      F8Ø41C          ; ungleich: RAM-Fehler ->
Ø414         move.l   d2,(aØ)+        ; Speicherinhalt wiederherstellen
Ø416         dbra     d3,F8Ø3FE       ; bis Endadresse wiederholen ->
Ø41A         clr.l    dØ              ; dØ := Ø = ok-Flag
Ø41C F8Ø41C  movem.l  (a7)+,d2-d3     ; Register wiederherstellen
Ø42Ø         jmp      (a5)            ; ---> Rücksprung
Ø422
Ø422 ;-----  Hardware-Test 1 (nicht gerufen)
Ø422
Ø422         move.w   #1FF,DFFØ96     ; DMACON: Bits Ø bis 8 löschen
Ø42A         move.w   #4ØØØ,DFFØ9A    ; INTENA: Interrupts sperren
Ø432         clr.l    dØ              ; dØ := Ø
Ø434         move.w   #1Ø1,dØ         ; dØ := 257 = Fehlerkode
Ø438         move.w   #7FFF,DFFØ9E    ; ADKCON: Alle Bits löschen
Ø44Ø         move.w   DFFØ1Ø,d4       ; d4 := ADKCONR
Ø446         cmpi.w   #Ø,d4           ; Alle Bits = Ø?
Ø44A         bne      F8Ø79Ø          ; Nein: Fehler ->
Ø44E         move.w   #E,d5           ; d5 := 14 = Wiederholungszähler
Ø452         move.w   #8ØØØ,d2        ; d2 := $8ØØØ = SET-Bit
Ø456         move.w   #1,d3           ; d3 := 1 = Testbit
Ø45A F8Ø45A  move.w   d3,d4           ; d4 := d3
Ø45C         add.w    d2,d3           ; Bit 31 einaddieren
Ø45E         move.w   d3,DFFØ9E       ; ADKCON: Bit (d4) setzen
Ø464         cmp.w    DFFØ1Ø,d4       ; d4 = ADKCONR?
Ø46A         bne      F8Ø79Ø          ; Nein: Fehler ->
Ø46E         lsl.w    #1,d3           ; Testbit um 1 nach links schieben
Ø47Ø         move.w   #7FFF,DFFØ9E    ; ADKCON: Alle Bits löschen
Ø478         dbra     d5,F8Ø45A       ; wiederholen, bis 15 Bits getestet ->
Ø47C         move.w   #1Ø2,dØ         ; dØ := 258 = Fehlerkode
Ø48Ø         move.w   #FFFF,DFFØ9E    ; ADKCON: Alle Bits setzen
Ø488         move.w   DFFØ1Ø,d4       ; d4 := ADKCONR
Ø48E         cmpi.w   #7FFF,d4        ; Bits Ø bis 14 gesetzt?
Ø492         bne      F8Ø79Ø          ; Nein: Fehler ->
Ø496         move.w   #E,d5           ; d5 := 14 = Wiederholungszähler
Ø49A         move.w   #1,d3           ; d3 := 1 = Testbit
Ø49E F8Ø49E  move.w   #7FFF,d4        ; d4 := Testmaske
Ø4A2         eor.w    d3,d4           ; Testbit löschen
Ø4A4         move.w   d3,DFFØ9E       ; ADKCON: Testbit löschen
Ø4AA         cmp.w    DFFØ1Ø,d4       ; ADKCONR = d4?
Ø4BØ         bne      F8Ø79Ø          ; Nein: Fehler ->
```

```
04B4          lsl.w    #1,d3            ; Testbit um 1 nach links schieben
04B6          move.w   #FFFF,DFF09E     ; ADKCON: Alle Bits setzen
04BE          dbra     d5,F8049E        ; wiederholen, bis 15 Bits getestet ->
04C2          move.w   #7FFF,DFF09E     ; ADKCON: Alle Bits löschen
04CA          move.w   #103,d0          ; d0 := 259 = Fehlerkode
04CE          move.w   DFF00E,d1        ; CLXDAT lesen und löschen
04D4          move.w   DFF00E,d1        ; d1 := CLXDAT
04DA          andi.w   #7FFF,d1         ; Bit 15 (unbenutzt) löschen
04DE          cmpi.w   #0,d1            ; Bits 0 bis 14 gelöscht?
04E2          bne      F80790           ; Nein: Fehler ->
04E6          move.w   #104,d0          ; d0 := 260 = Fehlerkode
04EA          move.w   #7FF,DFF096      ; DMACON: Bits 0 bis 10 löschen
04F2          move.w   DFF002,d4        ; d4 := DMACONR
04F8          andi.w   #7FF,d4          ; Bits 11 bis 15 löschen
04FC          cmpi.w   #0,d4            ; Alle Bits gelöscht?
0500          bne      F80790           ; Nein: Fehler ->
0504          move.w   #A,d5            ; d5 := 10 = Wiederholungszähler
0508          move.w   #8000,d2         ; d2 := $8000 = SET-Bit
050C          move.w   #1,d3            ; d3 := 1 = Testbit
0510 F80510   move.w   d3,d4            ; d4 := d3 = Testbit
0512          add.w    d2,d3            ; SET-Bit einaddieren
0514          move.w   d3,DFF096        ; DMACON: Testbit setzen
051A          move.w   DFF002,d1        ; d1 := DMACONR
0520          andi.w   #7FF,d1          ; Bits 11 bis 15 löschen
0524          cmp.w    d1,d4            ; Testbit gesetzt?
0526          bne      F80790           ; Nein: Fehler ->
052A          lsl.w    #1,d3            ; Testbit um 1 nach links schieben
052C          move.w   #7FF,DFF096      ; DMACON: Bits 0 bis 10 löschen
0534          dbra     d5,F80510        ; wiederholen, bis 11 Bits getestet ->
0538          move.w   #105,d0          ; d0 := 261 = Fehlerkode
053C          move.w   #87FF,DFF096     ; DMACON: Bits 0 bis 10 setzen
0544          move.w   DFF002,d4        ; d4 := DMACONR
054A          andi.w   #7FF,d4          ; Bits 11 bis 15 löschen
054E          cmpi.w   #7FF,d4          ; Bits 0 bis 10 gesetzt?
0552          bne      F80790           ; Nein: Fehler ->
0556          move.w   #A,d5            ; d5 := 10 = Wiederholungszähler
055A          move.w   #1,d3            ; d3 := 1 = Testbit
055E F8055E   move.w   #7FF,d4          ; d4: Bits 0 bis 10 setzen
0562          eor.w    d3,d4            ; Testbit löschen
0564          move.w   d3,DFF096        ; DMACON: Testbit löschen
```

```
056A          move.w  DFF002,d1          ; d1 := DMACONR
0570          andi.w  #7FF,d1            ; Bits 11 bis 15 löschen
0574          cmp.w   d1,d4              ; Testbit gelöscht?
0576          bne     F80790             ; Nein: Fehler ->
057A          lsl.w   #1,d3              ; Testbit um 1 nach links schieben
057C          move.w  #87FF,DFF096       ; DMACON: Bits 0 bis 10 setzen
0584          dbra    d5,F8055E          ; wiederholen, bis 11 Bits getestet ->
0588          move.w  #7FF,DFF096        ; DMACON: Bits 0 bis 10 löschen
0590          move.w  #106,d0            ; d0 := 262 = Fehlerkode
0594          move.w  #7FFF,DFF09A       ; INTENA: Alle Bits löschen
059C          move.w  #7FFF,DFF09C       ; INTREQ: Alle Bits löschen
05A4          move.w  DFF01E,d4          ; d4 := INTREQR
05AA          cmpi.w  #0,d4              ; Alle Bits gelöscht?
05AE          bne     F80790             ; Nein: Fehler ->
05B2          move.w  #A,d5              ; d5 := 10 = Wiederholungszähler
05B6          move.w  #8000,d2           ; d2 := $8000 = SET-Bit
05BA          move.w  #1,d3              ; d3 := 1 = Testbit
05BE F805BE   move.w  d3,d4              ; d4 := d3 = Testbit
05C0          add.w   d2,d3              ; SET-Bit einaddieren
05C2          move.w  d3,DFF09C          ; INTREQ: Testbit setzen
05C8          move.w  DFF01E,d1          ; d1 := INTREQR
05CE          cmp.w   d1,d4              ; Testbit gesetzt?
05D0          bne     F80790             ; Nein: Fehler ->
05D4 F805D4   lsl.w   #1,d3              ; Testbit um 1 nach links schieben
05D6          cmpi.w  #4,d3              ; Bit 2
05DA          beq.s   F805D4             ; übergehen
05DC          cmpi.w  #8,d3              ; Bit 3
05E0          beq.s   F805D4             ; übergehen
05E2          cmpi.w  #20,d3             ; Bit 5
05E6          beq.s   F805D4             ; übergehen
05E8          move.w  #7FFF,DFF09C       ; INTREQ: Alle Bits löschen
05F0          dbra    d5,F805BE          ; wiederholen, bis 11 Bits getestet ->
05F4          move.w  #107,d0            ; d0 := 263 = Fehlerkode
05F8          move.w  #BFFF,DFF09C       ; INTREQ: Bits 0 bis 13 setzen
0600          move.w  DFF01E,d4          ; d4 := INTREQR
0606          cmpi.w  #3FFF,d4           ; Bits 0 bis 13 gesetzt?
060A          bne     F80790             ; Nein: Fehler ->
060E          move.w  #A,d5              ; d5 := 10 = Wiederholungszähler
0612          move.w  #1,d3              ; d3 := 1 = Testbit
0616 F80616   move.w  #3FFF,d4           ; d4 := Testmaske
```

```
061A          eor.w    d3,d4            ; Testbit löschen
061C          move.w   d3,DFF09C        ; INTREQ: Testbit löschen
0622          move.w   DFF01E,d1        ; d1 := INTREQR
0628          cmp.w    d1,d4            ; Testbit gelöscht?
062A          bne      F80790           ; Nein: Fehler ->
062E          lsl.w    #1,d3            ; Testbit um 1 nach links schieben
0630          move.w   #BFFF,DFF09C     ; INTREQ: Bits 0 bis 13 löschen
0638          dbra     d5,F80616        ; wiederholen, bis 11 Bits getestet
063C          move.w   #7FFF,DFF09C     ; INTREQ: Alle Bits löschen
0644          move.w   #108,d0          ; d0 := 264 = Fehlerkode
0648          move.w   #7FFF,DFF09A     ; INTENA: Alle Bits löschen
0650          move.w   DFF01C,d4        ; d4 := INTENAR
0656          cmpi.w   #0,d4            ; Alle Bits gelöscht?
065A          bne      F80790           ; Nein: Fehler ->
065E          move.w   #D,d5            ; d5 := 13 = Wiederholungszähler
0662          move.w   #8000,d2         ; d2 := $8000 = SET-Bit
0666          move.w   #1,d3            ; d3 := 1 = Testbit
066A F8066A   move.w   d3,d4            ; d4 := d3 = Testbit
066C          add.w    d2,d3            ; SET-Bit einaddieren
066E          move.w   d3,DFF09A        ; INTENA: Testbit setzen
0674          move.w   DFF01C,d1        ; d1 := INTENAR
067A          cmp.w    d1,d4            ; Testbit gesetzt?
067C          bne      F80790           ; Nein: Fehler ->
0680          lsl.w    #1,d3            ; Testbit um 1 nach links schieben
0682          move.w   #7FFF,DFF09A     ; INTENA: Alle Bits löschen
068A          dbra     d5,F8066A        ; wiederholen, bis 14 Bits getestet ->
068E          move.w   #109,d0          ; d0 := 265 = Fehlerkode
0692          move.w   #BFFF,DFF09A     ; INTENA: Bits 0 bis 13 setzen
069A          move.w   DFF01C,d4        ; d4 := INTENAR
06A0          cmpi.w   #3FFF,d4         ; Bits 0 bis 13 gesetzt?
06A4          bne      F80790           ; Nein: Fehler ->
06A8          move.w   #D,d5            ; d5 := 13 = Wiederholungszähler
06AC          move.w   #1,d3            ; d3 := 1 = Testbit
06B0 F806B0   move.w   #3FFF,d4         ; d4 := Testmaske
06B4          eor.w    d3,d4            ; Testbit löschen
06B6          move.w   d3,DFF09A        ; INTENA: Testbit löschen
06BC          move.w   DFF01C,d1        ; d1 := INTENAR
06C2          cmp.w    d1,d4            ; Testbit gelöscht?
06C4          bne      F80790           ; Nein: Fehler ->
06C8          lsl.w    #1,d3            ; Testbit um 1 nach links schieben
```

```
06CA    move.w  #BFFF,DFF09A    ; INTENA: Bits 0 bis 13 setzen
06D2    dbra    d5,F806B0       ; wiederholen, bis 14 Bits getestet ->
06D6    move.w  #4000,DFF09A    ; INTENA: Alle Interrupts sperren
06DE    move.w  #FCFC,DFF036    ; JOYTEST: Alle Mauszähler setzen
06E6    move.w  DFF00A,d2       ; d2 := JOY0DAT
06EC    move.w  DFF00C,d3       ; d3 := JOY1DAT
06F2    move.w  #10B,d0         ; d0 := 267 = Fehlerkode
06F6    cmpi.w  #FCFC,d2        ; JOY0DAT ok?
06FA    bne     F80790          ; Nein: Fehler ->
06FE    move.w  #10C,d0         ; d0 := 268 = Fehlerkode
0702    cmpi.w  #FCFC,d3        ; JOY1DAT ok?
0706    bne     F80790          ; Nein: ->
070A    move.w  #0,DFF036       ; JOYTEST: Alle Mauszähler löschen
0712    move.w  DFF00A,d2       ; d2 := JOY0DAT
0718    move.w  DFF00C,d3       ; d3 := JOY1DAT
071E    move.w  #10D,d0         ; d0 := 269 = Fehlerkode
0722    cmpi.w  #0,d2           ; JOY0DAT ok?
0726    bne     F80790          ; Nein: Fehler ->
072A    move.w  #10E,d0         ; d0 := 270 = Fehlerkode
072E    cmpi.w  #0,d3           ; JOY1DAT ok?
0732    bne     F80790          ; Nein: Fehler ->
0736    move.w  #A8A8,DFF036    ; JOYTEST: Bitmuster 101010... schreiben
073E    move.w  DFF00A,d2       ; d2 := JOY0DAT
0744    move.w  DFF00C,d3       ; d3 := JOY1DAT
074A    move.w  #10F,d0         ; d0 := 271 = Fehlerkode
074E    cmpi.w  #A8A8,d2        ; JOY0DAT ok?
0752    bne     F80790          ; Nein: Fehler ->
0756    move.w  #110,d0         ; d0 := 272 = Fehlerkode
075A    cmpi.w  #A8A8,d3        ; JOY1DAT ok?
075E    bne     F80790          ; Nein: Fehler ->
0762    move.w  #5454,DFF036    ; JOYTEST: Bitmuster 010101... schreiben
076A    move.w  DFF00A,d2       ; d2 := JOY0DAT
0770    move.w  DFF00C,d3       ; d3 := JOY1DAT
0776    move.w  #111,d0         ; d0 := 273 = Fehlerkode
077A    cmpi.w  #5454,d2        ; JOY0DAT ok?
077E    bne     F80790          ; Nein: Fehler ->
0782    move.w  #112,d0         ; d0 := 274 = Fehlerkode
0786    cmpi.w  #5454,d3        ; JOY1DAT ok?
078A    bne     F80790          ; Nein: Fehler ->
078E    clr.l   d0              ; d0 := 0 = ok-Flag
```

```
0790 F80790  move.w  #1FF,DFF096         ; DMACON: Bits 0 bis 8 löschen
0798         move.w  #4000,DFF09A        ; INTENA: Alle Interrupts sperren
07A0         jmp     (a5)                ; ---> Rücksprung
07A2
07A2 ;-----  Audio-Test mit Melodie
07A2
07A2 F807A2  movem.l d2-d3/a2-a3,-(a7)   ; Register retten
07A6         link    a6,#0               ;
07AA         move.w  #1FF,DFF096         ; DMACON: Bits 0 bis 8 löschen
07B2         move.w  #8200,DFF096        ; DMACON: Bit 9 setzen
07BA         move.w  #4000,DFF09A        ; INTENA: Alle Interrupts sperren
07C2         clr.l   d2                  ; d2 := 0 = Wortzähler
07C4         lea     F80A14,a0           ; a0 -> Sample-Tabelle
07CA F807CA  move.w  -(a0),-(a7)         ; Sample-Daten in Stack kopieren
07CC         addq.w  #1,d2               ; Zähler inkrementieren
07CE         tst.w   (a0)                ; Ende der Tabelle?
07D0         bne.s   F807CA              ; Nein: ->
07D2         subq.w  #1,d2               ; Zähler um 1 erniedrigen
07D4         move.w  d2,-(a7)            ; und ebenfalls in Stack schreiben
07D6         movea.l a7,a2               ; a2 := a7 -> Sample-Daten im Stack
07D8         move.w  #0,d0               ; d0 := 0 = Audio-Kanal
07DC         move.w  #5,d3               ; d3 := 5 = Anstiegsdauer
07E0         move.w  #FF00,d2            ; d2 := $FF00 = Halte-Dauer 0.1 s
07E4         jsr     F8095A              ; ---> Audio-Adressen definieren
07EA         move.w  #CA,d1              ; d1 := 202 ergibt Frequenz 286 Hz
07EE         movea.l a2,a3               ; a3 := a2 -> Sample-Daten im Stack
07F0         jsr     F80974              ; ---> Audio-Parameter setzen
07F6         jsr     F80988              ; ---> Klang ausgeben
07FC         move.w  #3,d0               ; d0 := 3 = Audio-Kanal
0800         move.w  #5,d3               ; d3 := 5 = Anstiegsdauer
0804         move.w  #FFFF,d2            ; d2 := $FFFF = Haltedauer 0.1 s
0808         jsr     F8095A              ; ---> Audio-Adressen definieren
080E         move.w  #87,d1              ; d1 := 135 ergibt Frequenz 428 Hz
0812         movea.l a2,a3               ; a3 := a2 -> Sample-Daten im Stack
0814         jsr     F80974              ; ---> Audio-Parameter setzen
081A         jsr     F80988              ; ---> Klang ausgeben
0820         move.w  #0,d0               ; d0 := 0 = Audio-Kanal
0824         move.w  #5,d3               ; d3 := 5 = Anstiegsdauer
0828         move.w  #200,d2             ; d2 := $200 = Haltedauer 0.8 ms
082C         jsr     F8095A              ; ---> Audio-Adressen definieren
```

```
0832    move.w  #A0,d1      ; d1 := 160 ergibt Frequenz 361 Hz
0836    movea.l a2,a3       ; a3 := a2 -> Sample-Daten im Stack
0838    jsr     F80974      ; ---> Audio-Parameter setzen
083E    jsr     F80988      ; ---> Klang ausgeben
0844    move.w  #3,d0       ; d0 := 3 = Audio-Kanal
0848    move.w  #5,d3       ; d3 := 5 = Anstiegsdauer
084C    move.w  #FF00,d2    ; d2 := $FF00 = Haltedauer 0.1 s
0850    jsr     F8095A      ; ---> Audio-Adressen definieren
0856    move.w  #CA,d1      ; d1 := 202 ergibt Frequenz 286 Hz
085A    movea.l a2,a3       ; a3 := a2 -> Sample-Daten im Stack
085C    jsr     F80974      ; ---> Audio-Parameter setzen
0862    jsr     F80988      ; ---> Klang ausgeben
0868    move.w  #0,d0       ; d0 := 0 = Audio-Kanal
086C    move.w  #5,d3       ; d3 := 5 = Anstiegsdauer
0870    move.w  #FF00,d2    ; d2 := $FF00 = Haltedauer 0.1 s
0874    jsr     F8095A      ; ---> Audio-Adressen definieren
087A    move.w  #B4,d1      ; d1 := 180 ergibt Frequenz 321 Hz
087E    movea.l a2,a3       ; a3 := a2 -> Sample-Daten im Stack
0880    jsr     F80974      ; ---> Audio-Parameter setzen
0886    jsr     F80988      ; ---> Klang ausgeben
088C    move.w  #1,d0       ; d0 := 1 = Audio-Kanal
0890    move.w  #5,d3       ; d3 := 5 = Anstiegsdauer
0894    move.w  #FF00,d2    ; d2 := $FF00 = Haltedauer 0.1 s
0898    jsr     F8095A      ; ---> Audio-Adressen definieren
089E    move.w  #A0,d1      ; d1 := 160 ergibt Frequenz 361 Hz
08A2    movea.l a2,a3       ; a3 := a2 -> Sample-Daten im Stack
08A4    jsr     F80974      ; ---> Audio-Parameter setzen
08AA    jsr     F80988      ; ---> Klang ausgeben
08B0    move.w  #2,d0       ; d0 := 2 = Audio-Kanal
08B4    move.w  #5,d3       ; d3 := 5 = Anstiegsdauer
08B8    move.w  #FF00,d2    ; d2 := $FF00 = Haltedauer 0.1 s
08BC    jsr     F8095A      ; ---> Audio-Adressen definieren
08C2    move.w  #97,d1      ; d1 := 151 ergibt Frequenz 382 Hz
08C6    movea.l a2,a3       ; a3 := a2 -> Sample-Daten im Stack
08C8    jsr     F80974      ; ---> Audio-Parameter setzen
08CE    jsr     F80988      ; ---> Klang ausgeben
08D4    move.w  #1,d0       ; d0 := 1 = Audio-Kanal
08D8    move.w  #5,d3       ; d3 := 5 = Anstiegsdauer
08DC    move.w  #FF00,d2    ; d2 := $FF00 = Haltedauer 0.1 s
08E0    jsr     F8095A      ; ---> Audio-Adressen definieren
```

```
08E6        move.w  #A0,d1              ; d1 := 160 ergibt Frequenz 361 Hz
08EA        movea.l a2,a3               ; a3 := a2 -> Sample-Daten im Stack
08EC        jsr     F80974              ; ---> Audio-Parameter setzen
08F2        jsr     F80988              ; ---> Klang ausgeben
08F8        move.w  #2,d0               ; d0 := 2 = Audio-Kanal
08FC        move.w  #5,d3               ; d3 := 5 = Anstiegsdauer
0900        move.w  #FF00,d2            ; d2 := $FF00 = Haltedauer 0.1 s
0904        jsr     F8095A              ; ---> Audio-Adressen definieren
090A        move.w  #B4,d1              ; d1 := 180 ergibt Frequenz 321 Hz
090E        movea.l a2,a3               ; a3 := a2 -> Sample-Daten im Stack
0910        jsr     F80974              ; ---> Audio-Parameter setzen
0916        jsr     F80988              ; ---> Klang ausgeben
091C        move.w  #1,d0               ; d0 := 1 = Audio-Kanal
0920        move.w  #5,d3               ; d3 := 5 = Anstiegsdauer
0924        move.w  #FF00,d2            ; d2 := $FF00 = Haltedauer 0.1 s
0928        jsr     F8095A              ; ---> Audio-Adressen definieren
092E        move.w  #B4,d1              ; d1 := 180 ergibt Frequenz 321 Hz
0932        movea.l a2,a3               ; a3 := a2 -> Sample-Daten im Stack
0934        jsr     F80974              ; ---> Audio-Parameter setzen
093A        jsr     F80988              ; ---> Klang ausgeben
0940        move.w  #1FF,DFF096         ; DMACON: Bits 0 bis 8 löschen
0948        move.w  #4000,DFF09A        ; INTENA: Interrupts sperren
0950        clr.l   d0                  ; d0 := 0
0952        unlk    a6                  ; Stackbereich auflösen
0954        movem.l (a7)+,d2-d3/a2-a3   ; Register wiederherstellen
0958        jmp     (a5)                ; ---> Rücksprung
095A
095A ;----- Audio-Adressen definieren
095A
095A F8095A move.w  d0,d1               ; d1 := d0 = Audio-Kanal-Nummer
095C        move.w  #10,d0              ; d0 := 16
0960        mulu    d1,d0               ; d0 := 16*Kanalnummer
0962        movea.l #DFF0A0,a0          ; a0 -> AUD0LC: Kanal 0 Datenzeiger
0968        lea     0(a0,d0.w),a0       ; a0 -> Kanal (d0) Datenzeiger
096C        move.w  #1,d0               ; d0 := 1
0970        lsl.w   d1,d0               ; d0 := Kanalnummer-Bit
0972        rts
0974
0974 ;----- Audio-Parameter setzen
0974
```

```
0974 F80974  move.w  (a3)+,4(a0)       ; AUDxLEN := Zahl der Sample-Worte
0978         move.l  a3,0(a0)          ; AUDxLOC := Anfang der Audio-Daten
097C         move.w  d1,6(a0)          ; AUDxPER := Sampling-Periode
0980         move.w  #0,8(a0)          ; AUDxVOL := 0
0986         rts
0988
0988 ;-----  Klang ausgeben
0988
0988 F80988  move.l  d0,-(a7)          ; d0 (Kanalnummer) retten
098A         ori.w   #8000,d0          ; SET-Bit setzen
098E         move.w  d0,DFF096         ; DMACON: Audio Channel DMA Enable
0994         move.w  d3,d1             ; d1 := d3 = Anstiegsdauer
0996         subq.w  #1,d1             ; um 1 vermindern
0998         movea.w #0,a1             ; a1 := 0 = Lautstärke
099C F8099C  move.w  a1,8(a0)          ; AUDxVOL := a1
09A0         addq.w  #1,a1             ; Lautstärke erhöhen
09A2         move.w  #20,d0            ; Zähler für 50 us setzen
09A6 F809A6  dbra    d0,F809A6         ; 50 us warten ->
09AA         dbra    d1,F8099C         ; wiederholen, bis d1 = -1 ->
09AE F809AE  dbra    d2,F809AE         ; Haltezeit warten ->
09B2         move.w  d3,d1             ; d1 := d3 = Anstiegszeit
09B4 F809B4  move.w  d1,8(a0)          ; AUDxVOL := d1
09B8         move.w  #80,d0            ; Zähler für 200 us setzen
09BC F809BC  dbra    d0,F809BC         ; 200 us warten ->
09C0         dbra    d1,F809B4         ; wiederholen, bis d1 = -1 ->
09C4         move.l  (a7)+,d0          ; d0 (Kanalnummer) wiederherstellen
09C6         andi.w  #F,d0             ; SET-Bit löschen
09CA         move.w  d0,DFF096         ; DMACON: Audio DMA abschalten
09D0         rts
09D2
09D2 ;-----  Sample-Tabelle für Sinuskurve
09D2
09D2         DC.W    0
09D4         DC.B    0,0D,19,25,31,3C,47,51,5B,63,6A,71,76,7A,7E,7F
09E4         DC.B    7F,7F,7E,7A,76,71,6A,63,5B,51,47,3C,31,25,19,0D
09F4         DC.B    0,F3,E7,DB,CF,C4,B9,AF,A5,9D,96,8F,8A,86,82,81
0A04         DC.B    81,81,82,86,8A,8F,96,9D,A5,AF,B9,C4,CF,DB,E7,F3
0A14 F80A14  DC.L    0
0A18
0A18 ;-----  Disk-Boot
```

```
ØA18
ØA18 F8ØA18  link     a6,#-14           ; 2Ø Bytes auf Stack reservieren
ØA1C         movem.l d2-d5/a2,-(a7)     ; Register retten
ØA2Ø         clr.b    -1(a6)            ; Graphik-Ausgabeflag löschen
ØA24         move.l   #1ØØØØ,-(a7)      ; Anfang des Disk-Lesepuffers
ØA2A         clr.l    -(a7)             ; Laufwerknummer = Ø
ØA2C         pea      -14(a6)           ; Adresse des Stackrahmens
ØA3Ø         jsr      F8ØBEØ            ; ---> Disklesen vorbereiten
ØA36         move.l   #FØØØ,d5          ; d5 := Anfang der Copperliste
ØA3C         move.l   d5,-(a7)          ; auf Stack
ØA3E         jsr      F811AC            ; ---> Graphikausgabe vorbereiten
ØA44         move.l   #FØØØØØ,d2
ØA4A         cmpi.l   #F8ØØØØ,d2        ; $F8ØØØØ = $FØØØØØ?
ØA5Ø         lea      1Ø(a7),a7         ; Stackzeiger korrigieren
ØA54         bne.s    F8ØA64            ; Nicht gleich (d.h.: immer): ->
ØA56         move.l   #F4ØØØØ,d4
ØA5C         move.l   #F4ØØØ4,dØ
ØA62         bra.s    F8ØA7Ø            ; --->
ØA64
ØA64 F8ØA64  move.l   #FCØØØØ,d4        ; d4 := Anfang des Kickstart-RAM
ØA6A         move.l   #FCØØØ4,dØ        ; dØ := Startadresse von Exec
ØA7Ø F8ØA7Ø  pea      -14(a6)           ; Adresse des Stackrahmens
ØA74         jsr      F8ØE94            ; ---> Motor einschalten, 1 s warten
ØA7A         pea      -14(a6)           ; Adresse des Stackrahmens
ØA7E         jsr      F8ØC4E            ; ---> Lesekopf auf Spur Ø setzen
ØA84         bclr.b   #6,-13(a6)
ØA8A         movea.l  #FCØØØØ,a2        ; a2 := Anfang des Kickstart-RAM
ØA9Ø         clr.l    -(a7)             ; Sektornummer Ø auf Stack
ØA92         move.l   a2,-(a7)          ; Zieladresse für Einlesen
ØA94         pea      -14(a6)           ; Adresse des Stackrahmens
ØA98         jsr      F8189Ø            ; ---> Sektor Ø in Speicher lesen
ØA9E         move.l   dØ,d3             ; d3 := dØ = Fehlerkode
ØAAØ         lea      14(a7),a7         ; Stackzeiger korrigieren
ØAA4         bne      F8ØAE8            ; Fehler ist aufgetreten: ->
ØAA8         moveq    #Ø,d2             ; d2 := Ø
ØAAA F8ØAAA  move.w   d2,dØ             ; dØ := d2 = Vergleichszeiger
ØAAC         movea.l  #F8ØBDC,aØ        ; aØ -> 'KICK'
ØAB2         move.b   Ø(aØ,dØ.w),dØ     ; dØ.b := Zeichen aus 'KICK'
ØAB6         move.w   d2,d1             ; d1 := d2 = Vergleichszeiger
ØAB8         movea.l  a2,aØ             ; aØ -> Anfang des gelesenen Sektors
```

```
0ABA          cmp.b    0(a0,d1.w),d0       ; Steht dort auch 'KICK'?
0ABE          bne      F80AE8              ; Nein: Falsche Diskette ->
0AC2          addq.l   #1,d2               ; Vergleichszeiger erhöhen
0AC4          moveq    #4,d0               ; d0 := Länge des Wortes 'KICK'
0AC6          cmp.l    d2,d0               ; Ende des Wortes erreicht?
0AC8          bgt.s    F80AAA              ; Nein: Vergleich fortsetzen ->
0ACA          move.l   #FC0000,d4          ; d4 := Zieladresse für Disklesen
0AD0          moveq    #1,d2               ; d2 := Sektornummer
0AD2 F80AD2   move.l   d2,-(a7)            ; Sektornummer und
0AD4          move.l   d4,-(a7)            ; Zieladresse auf Stack
0AD6          pea      -14(a6)             ; Adresse des Stackrahmens
0ADA          jsr      F81890              ; ---> Sektor in Speicher lesen
0AE0          move.l   d0,d3               ; d3 := d0 = Fehlerkode
0AE2          lea      C(a7),a7            ; Stackzeiger korrigieren
0AE6          beq.s    F80B0A              ; Kein Fehler aufgetreten: ->
0AE8 F80AE8   pea      -14(a6)             ; Adresse des Stackrahmens
0AEC          jsr      F80EC8              ; ---> Motor ausschalten
0AF2          move.l   d5,-(a7)            ; d5 -> Anfang der Copperliste
0AF4          pea      -1(a6)              ; Adresse des Graphik-Ausgabeflags
0AF8          pea      -14(a6)             ; Adresse des Stackrahmens
0AFC          jsr      F80B38              ; ---> Graphik ausgeben, Diskwechsel
0B02          lea      10(a7),a7           ; Stackzeiger korrigieren
0B06          bra      F80A70              ; ---> neuer Einleseversuch
0B0A
0B0A F80B0A   addi.l   #200,d4             ; Zieladresse um Sektorlänge erhöhen
0B10          addq.l   #1,d2               ; Sektornummer um 1 erhöhen
0B12          cmpi.l   #200,d2             ; Schon 512 Sektoren eingelesen?
0B18          ble.s    F80AD2              ; Nein: Einlesen fortsetzen ->
0B1A          pea      -14(a6)             ; Adresse des Stackrahmens
0B1E          jsr      F80EC8              ; ---> Motor ausschalten
0B24          move.w   #7FFF,DFF096        ; DMACON: Alle Bits löschen
0B2C          moveq    #0,d0               ; d0 := 0
0B2E          addq.l   #4,a7               ; Stackzeiger korrigieren
0B30          movem.l  (a7)+,d2-d5/a2      ; Register wiederherstellen
0B34          unlk     a6                  ; Stackrahmen freigeben
0B36          rts
0B38
0B38 ;-----   Graphik ausgeben, auf Diskwechsel warten
0B38
0B38 F80B38   movem.l  d2-d4/a2,-(a7)      ; Register retten
```

```
ØB3C         move.l  14(a7),d3        ; d3 := Adresse des Stackrahmens
ØB4Ø         movea.l 18(a7),a2        ; a2 -> Graphik-Ausgabeflag
ØB44         move.l  1C(a7),d2        ; d2 -> Anfang der Copperliste
ØB48         move.w  #1ØØ,d4          ; d4 := Bit Plane DMA Enable Bit
ØB4C         move.l  d3,-(a7)         ; Adresse des Stackrahmens übergeben
ØB4E         jsr     F8ØD8Ø           ; ---> Laufwerk Ø anwählen
ØB54         cmpi.b  #1,(a2)          ; Graphik-Ausgabeflag gesetzt?
ØB58         addq.l  #4,a7            ; Stackzeiger korrigieren
ØB5A         beq.s   F8ØB6A           ; Graphik-Ausgabeflag gesetzt: ->
ØB5C         move.l  d2,-(a7)         ; Anfang der Copperliste Übergeben
ØB5E         jsr     F813AØ           ; ---> Farben setzen, BitMap erzeugen
ØB64         move.b  #1,(a2)          ; Graphik-Ausgabeflag setzen
ØB68         addq.l  #4,a7            ; Stackzeiger korrigieren
ØB6A F8ØB6A  jsr     F813D6           ; ---> Warten auf Bildwechsel
ØB7Ø         move.w  d4,d2            ; d2 := d4 = Bit Plane DMA Enable Bit
ØB72         ori.w   #8ØØØ,d2         ; SET-Bit einodern
ØB76         move.w  d2,DFFØ96        ; DMACON: Bit Plane DMA freigeben
ØB7C F8ØB7C  moveq   #Ø,dØ            ; dØ := Ø
ØB7E         move.b  BFEØØ1,dØ        ; dØ := CIAA PRA
ØB84         moveq   #%11111Ø11,d1    ; DSKCHANGE-Bit := Ø
ØB86         or.l    d1,dØ            ; Alle anderen Bits auf 1 setzen
ØB88         moveq   #FF,d1           ; Alle Bits in d1 setzen
ØB8A         cmp.l   dØ,d1            ; War DSKCHANGE-Bit gesetzt?
ØB8C         sne     d2               ; Wenn nicht, dann d2 := -1
ØB8E         neg.b   d2               ; d2 := 1, wenn keine Disk im Laufwerk
ØB9Ø         beq.s   F8ØB7C           ; sonst warten, bis Disk entnommen ->
ØB92 F8ØB92  moveq   #Ø,d2            ; d2 := Ø
ØB94         move.b  BFEØØ1,d2        ; d2 := CIAA PRA
ØB9A         moveq   #%11111Ø11,dØ    ; DSKCHANGE-Bit := Ø
ØB9C         or.l    dØ,d2            ; Alle anderen Bits in d2 setzen
ØB9E         moveq   #FF,d1           ; Alle Bits in d1 setzen
ØBAØ         cmp.l   d2,d1            ; Disk im Laufwerk?
ØBA2         beq.s   F8ØBC4           ; Ja: ->
ØBA4         move.l  #7A12Ø,-(a7)     ; Wartezahl 5ØØØØØ: 1 s
ØBAA         jsr     F8ØDB4           ; ---> warten
ØBBØ         eori.b  #2,BFD1ØØ        ; CIAB PRB DSKDIREC-Bit flippen
ØBB8         move.l  d3,-(a7)         ; Adresse des Stackrahmens übergeben
ØBBA         jsr     F8ØDD4           ; ---> Kopfschritt (DSKCHANGE-Update)
ØBCØ         addq.l  #8,a7            ; Stackzeiger korrigieren
ØBC2         bra.s   F8ØB92           ; ---> warten, bis Disk im Laufwerk
```

```
0BC4
0BC4 F80BC4  move.l  d3,-(a7)          ; Adresse des Stackrahmens übergeben
0BC6         jsr     F80DA6            ; ---> Laufwerk abwählen
0BCC         move.w  d4,DFF096         ; DMACON: Bit Plane DMA sperren
0BD2         addq.l  #4,a7             ; Stackzeiger korrigieren
0BD4         movem.l (a7)+,d2-d4/a2    ; Register wiederherstellen
0BD8         rts
0BDA
0BDA         DC.W    0
0BDC
0BDC F80BDC  DC.B    'KICK'            ; Kennwort am Anfang von Sektor 0
0BE0
0BE0 ;-----  Disklesen vorbereiten
0BE0
0BE0 F80BE0  movem.l d2/a2-a3,-(a7)    ; Register retten
0BE4         movea.l 10(a7),a0         ; a0 := Adresse des Stackrahmens
0BE8         move.l  14(a7),d1         ; d1 := Laufwerknummer
0BEC         move.l  18(a7),d2         ; d2 := Adresse des Lesepuffers
0BF0         movea.l #DFF000,a3        ; a3 -> ChipBase
0BF6         movea.l #F80EF0,a1        ; a1 -> Initialisierungstabelle
0BFC         movea.l a0,a2             ; a2 := a0 = Adresse des Stackrahmens
0BFE         moveq   #11,d0            ; d0 := 0: Initialisierungszähler
0C00 F80C00  move.b  (a1)+,(a2)+       ; Stackrahmen initialisieren
0C02         dbra    d0,F80C00         ; mit 18 Nullen
0C06         andi.b  #%11000011,BFE201 ; CIAA DDRA: Bits 2 bis 5 = Input
0C0E         move.b  #FF,BFD100        ; CIAB PRB: Alle Bits setzen
0C16         move.b  #FF,BFD300        ; CIAB DDRB: Alle Bits = Output
0C1E         moveq   #1,d0             ; d0 := 1
0C20         addq.b  #3,d1             ; d1 := DiskSelect-Bit-Nummer
0C22         asl.l   d1,d0             ; DiskSelect-Bit in d0 setzen
0C24         move.b  d0,(a0)           ; und im Stackrahmen abspeichern
0C26         move.l  d2,6(a0)          ; ebenso die Adresse des Lesepuffers
0C2A         move.w  #7FFF,9A(a3)      ; INTENA: Alle Interrupts sperren
0C30         move.w  #8210,96(a3)      ; DMACON: DMAEN und DSKEN setzen
0C36         move.w  #7F00,9E(a3)      ; ADKCON: Bits 8 bis 14 löschen
0C3C         move.w  #8500,9E(a3)      ; ADKCON: WORDSYNC und FAST setzen
0C42         move.w  #4489,7E(a3)      ; DSKSYNC := Synchron-Wort
0C48         movem.l (a7)+,d2/a2-a3    ; Register wiederherstellen
0C4C         rts
0C4E
```

```
ØC4E ;-----  Lesekopf auf Außenspur (Spur Ø) setzen
ØC4E
ØC4E F8ØC4E  movem.l a2-a3,-(a7)        ; Register retten
ØC52         movea.l C(a7),a2           ; a2 := Adresse des Stackrahmens
ØC56         movea.l #BFD1ØØ,a3         ; a3 -> CIAB PRB
ØC5C         move.l  a2,-(a7)           ; Adresse des Stackrahmens übergeben
ØC5E         jsr     F8ØD8Ø             ; ---> Laufwerk Ø anwählen
ØC64         clr.w   4(a2)              ; Spurnummer Ø abspeichern
ØC68         ori.b   #4,(a3)            ; DSKSIDE := 1 (unten)
ØC6C         andi.b  #FD,(a3)           ; DSKDIREC := Ø (Kopfschritt nach innen)
ØC7Ø         addq.l  #4,a7              ; Stackzeiger korrigieren
ØC72 F8ØC72  moveq   #Ø,dØ              ; dØ := Ø
ØC74         move.b  BFEØØ1,dØ          ; dØ := CIAA PRA
ØC7A         moveq   #%111Ø1111,d1      ; DSKTRACKØ-Bit := Ø
ØC7C         or.l    d1,dØ              ; Alle anderen Bits setzen
ØC7E         moveq   #FF,d1             ; Alle Bits in d1 setzen
ØC8Ø         cmp.l   dØ,d1              ; Lesekopf auf Spur Ø?
ØC82         beq.s   F8ØC9Ø             ; Nein: ->
ØC84         move.l  aØ,-(a7)           ;
ØC86         jsr     F8ØDD4             ; ---> Kopfschritt ausführen
ØC8C         addq.l  #4,a7              ; Stackzeiger korrigieren
ØC8E         bra.s   F8ØC72             ; ---> wiederholen
ØC9Ø
ØC9Ø F8ØC9Ø  ori.b   #2,(a3)            ; DSKDIREC := 1 (Kopfschritt nach außen)
ØC94 F8ØC94  moveq   #Ø,dØ              ; dØ := Ø
ØC96         move.b  BFEØØ1,dØ          ; dØ := CIAA PRA
ØC9C         moveq   #%111Ø1111,d1      ; DSKTRACKØ-Bit := Ø
ØC9E         or.l    d1,dØ              ; Alle anderen Bits setzen
ØCAØ         moveq   #FF,d1             ; Alle Bits von d1 setzen
ØCA2         cmp.l   dØ,d1              ; Lesekopf auf Spur Ø?
ØCA4         bne     F8ØCB4             ; Ja: ->
ØCA8         move.l  a2,-(a7)           ;
ØCAA         jsr     F8ØDD4             ; ---> Kopfschritt ausführen
ØCBØ         addq.l  #4,a7              ; Stackzeiger korrigieren
ØCB2         bra.s   F8ØC94             ; ---> wiederholen
ØCB4
ØCB4 F8ØCB4  pea     3A98               ; Wartezahl 15ØØØ: 3Ø ms
ØCB8         jsr     F8ØDB4             ; ---> warten
ØCBE         move.l  a2,-(a7)
ØCCØ         jsr     F8ØDA6             ; ---> Laufwerk abwählen
```

```
0CC6        addq.l  #8,a7             ; Stackzeiger korrigieren
0CC8        movem.l (a7)+,a2-a3       ; Register wiederherstellen
0CCC        rts
0CCE
0CCE ;----- Lesekopf auf bestimmte Spur setzen
0CCE
0CCE F80CCE movem.l d2-d6/a2-a3,-(a7) ; Register retten
0CD2        movea.l 20(a7),a2         ; a2 := Adresse des Stackrahmens
0CD6        move.l  24(a7),d2         ; d2 := Spurnummer
0CDA        moveq   #0,d0             ; d0 := 0
0CDC        movea.l #BFD100,a3        ; a3 -> CIAB PRB
0CE2        move.l  d2,d6             ; d6 := d2 = Spurnummer
0CE4        moveq   #1,d0             ; d0 := 1
0CE6        and.l   d0,d6             ; d6 := Spurnummer modulo 2 = Seite
0CE8        move.l  d2,d4             ; d4 := Spurnummer
0CEA        asr.l   #1,d4             ; d4 := Spurnummer Ö 2 = Zylinder
0CEC        move.w  4(a2),d5          ; d5 := alte Spurnummer
0CF0        ext.l   d5                ; auf Langwort erweitern
0CF2        moveq   #1,d0             ; d0 := 1
0CF4        and.l   d0,d5             ; d5 := alte Spurnummer modulo 2 = Seite
0CF6        move.w  4(a2),d3          ; d3 := alte Spurnummer
0CFA        ext.l   d3                ; auf Langwort erweitern
0CFC        asr.l   #1,d3             ; d3 := alte Spurnummer Ö 2 = Zylinder
0CFE        move.w  4(a2),d0          ; d0 := alte Spurnummer
0D02        ext.l   d0                ; auf Langwort erweitern
0D04        cmp.l   d2,d0             ; Neue Spurnummer = alte Spurnummer?
0D06        beq     F80D7A            ; Ja: fertig ->
0D0A        move.l  a2,-(a7)          ; Adresse des Stackrahmens übergeben
0D0C        jsr     F80D80            ; ---> Laufwerk 0 anwählen
0D12        cmp.l   d6,d5             ; Neue Seite = alte Seite?
0D14        addq.l  #4,a7             ; Stackzeiger korrigieren
0D16        beq.s   F80D32            ; Gleiche Seite: ->
0D18        tst.l   d6                ; Neue Seite 'unten'?
0D1A        beq.s   F80D22            ; Ja: ->
0D1C        andi.b  #%11111011,(a3)   ; DSKSIDE-Bit := 0 ('oben')
0D20        bra.s   F80D26            ; --->
0D22
0D22 F80D22 ori.b   #4,(a3)           ; DSKSIDE-Bit := 1 ('unten')
0D26 F80D26 pea     C8                ; Wartezahl 200: 0.4 ms
0D2A        jsr     F80DB4            ; ---> warten
```

```
ØD3Ø        addq.l  #4,a7              ; Stackzeiger korrigieren
ØD32 F8ØD32 cmp.l   d4,d3              ; Neuer Zylinder = alter Zylinder?
ØD34        beq.s   F8ØD6C             ; Ja: fertig ->
ØD36        cmp.l   d4,d3              ; Neuer Zylinder > alter Zylinder?
ØD38        bge.s   F8ØD44             ; Nein: ->
ØD3A        andi.b  #%11111101,(a3)    ; DSKDIREC := Ø: Kopfschritt nach innen
ØD3E        move.l  d4,d5              ; d5 := d4 = neuer Zylinder
ØD4Ø        sub.l   d3,d5              ; d5 := Zahl der Kopfschritte
ØD42        bra.s   F8ØD4C             ; --->
ØD44
ØD44 F8ØD44 ori.b   #2,(a3)            ; DSKDIREC := 1: Kopfschritt nach außen
ØD48        move.l  d3,d5              ; d5 := alter Zylinder
ØD4A        sub.l   d4,d5              ; d5 := Zahl der Kopfschritte
ØD4C F8ØD4C move.l  d5,dØ              ; dØ := Zahl der Kopfschritte
ØD4E        subq.l  #1,d5              ; um 1 vermindern
ØD5Ø        move.l  dØ,dØ              ; dØ = Ø?
ØD52        beq.s   F8ØD6Ø             ; Ja: fertig ->
ØD54        move.l  a2,-(a7)           ; Adresse des Stackrahmens übergeben
ØD56        jsr     F8ØDD4             ; ---> Kopfschritt ausführen
ØD5C        addq.l  #4,a7              ; Stackzeiger korrigieren
ØD5E        bra.s   F8ØD4C             ; ---> wiederholen
ØD6Ø
ØD6Ø F8ØD6Ø pea     3A98               ; Wartezahl 15ØØØ: 3Ø ms
ØD64        jsr     F8ØDB4             ; ---> warten
ØD6A        addq.l  #4,a7              ; Stackzeiger korrigieren
ØD6C F8ØD6C move.w  d2,4(a2)           ; Neue Spurnummer abspeichern
ØD7Ø        move.l  a2,-(a7)           ; Adresse des Stackrahmens übergeben
ØD72        jsr     F8ØDA6             ; ---> Laufwerke abwählen
ØD78        addq.l  #4,a7              ; Stackzeiger korrigieren
ØD7A F8ØD7A movem.l (a7)+,d2-d6/a2-a3  ; Register wiederherstellen
ØD7E        rts
ØD8Ø
ØD8Ø ;----- Laufwerk anwählen, ggf. Motor ein- oder ausschalten
ØD8Ø
ØD8Ø F8ØD8Ø movea.l 4(a7),aØ           ; aØ -> Stackrahmen
ØD84        movea.l #BFD1ØØ,a1         ; a1 -> CIAB PRB
ØD8A        ori.b   #8Ø,(a1)           ; DSKMOTOR-Bit := 1 ('aus')
ØD8E        btst.b  #7,1(aØ)           ; Motorflag testen
ØD94        sne     dØ                 ; dØ := -1, wenn Motorflag gesetzt
ØD96        neg.b   dØ                 ; dØ := 1, wenn Motorflag gesetzt
```

```
ØD98          lsl.b    #7,dØ          ; Bit Ø nach Bit 7 verschieben
ØD9A          not.b    dØ             ; und Bit flippen
ØD9C          and.b    dØ,(a1)        ; DSKMOTOR-Bit := Ø ('ein') wenn Motorfl
ØD9E          move.b   (aØ),dØ        ; dØ := LaufwerkSelect-Bit
ØDAØ          not.b    dØ             ; flippen
ØDA2          and.b    dØ,(a1)        ; DSKSELx-Bit := Ø wählt Laufwerk an
ØDA4          rts
ØDA6
ØDA6 ;-----   Alle Laufwerke abwählen
ØDA6
ØDA6 F8ØDA6   move.l   4(a7),dØ       ; dØ := Adresse des Stackrahmens
ØDAA          ori.b    #78,BFD1ØØ     ; Alle DSKSEL-Bits setzen
ØDB2          rts
ØDB4
ØDB4 ;-----   Warteroutine
ØDB4
ØDB4 F8ØDB4   move.l   d2,-(a7)       ; d2 retten
ØDB6          move.l   8(a7),d2       ; d2 := Wartezahl n
ØDBA          move.l   d2,dØ          ; dØ := d2 = n
ØDBC          asr.l    #2,dØ          ; dØ := n/4
ØDBE          move.l   dØ,d2          ; d2 := n/4
ØDCØ F8ØDCØ   subq.l   #1,d2          ; d2 dekrementieren
ØDC2          beq.s    F8ØDCC         ; d2 = Ø: ->
ØDC4          jsr      F8ØDDØ         ; ---> Zeit verbrauchen
ØDCA          bra.s    F8ØDCØ         ; ---> wiederholen
ØDCC
ØDCC F8ØDCC   move.l   (a7)+,d2       ; d2 wiederherstellen
ØDCE          rts
ØDDØ
ØDDØ F8ØDDØ   moveq    #Ø,dØ
ØDD2          rts
ØDD4
ØDD4 ;-----   Kopfschritt ausführen
ØDD4
ØDD4 F8ØDD4   move.l   4(a7),dØ       ; dØ := Adresse des Stackrahmens
ØDD8          andi.b   #%1111111Ø,BFD1ØØ ; DSKSTEP-Bit := Ø
ØDEØ          ori.b    #1,BFD1ØØ      ; DSKSTEP-Bit := 1
ØDE8          pea      BB8            ; Wartezahl 3ØØØ: 6 ms
ØDEC          jsr      -3A(pc)        ; ---> F8ØDB4: warten
ØDFØ          addq.w   #4,(a7)+       ; Stackzeiger korrigieren
```

```
ØDF2         rts
ØDF4
ØDF4 ;-----  Spur von Disk einlesen
ØDF4
ØDF4 F8ØDF4  movem.l d2-d6/a2-a3,-(a7)  ; Register retten
ØDF8         movea.l 2Ø(a7),a2          ; a2 -> Stackrahmen
ØDFC         move.l  #36F2,d6           ; d6 := 14Ø66 = Wert für DSKLEN
ØEØ2         moveq   #Ø,d4              ; d4 := Ø = Rückmeldekode 'ok'
ØEØ4         moveq   #Ø,dØ              ; dØ := Ø (nicht sinnvoll, vgl. ØE44)
ØEØ6         move.l  #3ØD4Ø,d5          ; d5 := 2ØØØØØ für Timeout nach 1.3 s
ØEØC         movea.l #DFFØØØ,a3         ; a3 -> ChipBase
ØE12         movea.l 6(a2),aØ           ; aØ -> Lesepuffer
ØE16         addq.l  #6,aØ              ; aØ um 6 erhöhen
ØE18         move.l  aØ,d2              ; d2 := aØ
ØE1A         move.w  #2,9C(a3)          ; INTREQ: DSKBKL-Bit löschen
ØE2Ø         move.l  d2,2Ø(a3)          ; DSKPTH: Disk-Lesezeiger setzen
ØE24         move.l  a2,-(a7)           ; Adresse des Stackrahmens übergeben
ØE26         jsr     -A8(pc)            ; ---> F8ØD8Ø: Laufwerk anwählen
ØE2A         moveq   #Ø,d2              ; d2 := Ø
ØE2C         move.b  BFEØØ1,d2          ; d2 := CIAA PRA
ØE32         moveq   #%11111Ø11,d3      ; DSKCHANGE-Bit := Ø
ØE34         or.l    d3,d2              ; Alle anderen Bits in d2 setzen
ØE36         moveq   #FF,d3             ; Alle Bits in d3 setzen
ØE38         cmp.l   d2,d3              ; DSKCHANGE-Bit gesetzt?
ØE3A         addq.l  #4,a7              ; Stackzeiger korrigieren
ØE3C         bne     F8ØE82             ; Keine Disk im Laufwerk: ->
ØE4Ø         clr.w   24(a3)             ; DSKLEN := Ø
ØE44         move.l  d6,dØ              ; dØ := d6 = 14Ø66 = Zahl der Bytes
ØE46         asr.l   #1,dØ              ; dØ := 7Ø33 = Zahl der Worte
ØE48         ori.l   #8ØØØ,dØ           ; DMAEN-Bit setzen
ØE4E         move.w  dØ,24(a3)          ; DSKLEN setzen
ØE52         move.w  dØ,24(a3)          ; Disk-DMA starten
ØE56 F8ØE56  subq.l  #1,d5              ; Schleifenzähler dekrementieren
ØE58         bge.s   F8ØE6Ø             ; noch nicht negativ: ->
ØE5A         moveq   #16,d2             ; d2 := 22 = Fehlerkode
ØE5C         move.l  d2,dØ              ; dØ := Fehlerkode
ØE5E         bra.s   F8ØE8E             ; ---> Abschluß
ØE6Ø
ØE6Ø F8ØE6Ø  moveq   #Ø,dØ              ; dØ := Ø
ØE62         move.w  1E(a3),dØ          ; dØ := INTREQR
```

```
ØE66        btst.l  #1,dØ               ; DSKBLK-Bit gesetzt?
ØE6A        beq.s   F8ØE56              ; Nein: warten ->
ØE6C        clr.w   24(a3)              ; DSKLEN löschen
ØE7Ø        moveq   #Ø,d2               ; d2 := Ø
ØE72        move.b  BFEØØ1,d2           ; d2 := CIAA PRA
ØE78        moveq   #%11111Ø11,d3       ; DSKCHANGE-Bit := Ø
ØE7A        or.l    d3,d2               ; Alle anderen Bits in d2 setzen
ØE7C        moveq   #FF,d3              ; Alle Bits in d3 setzen
ØE7E        cmp.l   d2,d3               ; Disk im Laufwerk?
ØE8Ø        beq.s   F8ØE84              ; Ja: ->
ØE82 F8ØE82 moveq   #19,d4              ; d4 := 25 = Fehlerkode
ØE84 F8ØE84 move.l  a2,-(a7)            ; Adresse des Stackrahmens übergeben
ØE86        jsr     -E2(pc)             ; ---> F8ØDA6: Laufwerke abwählen
ØE8A        move.l  d4,dØ               ; dØ := d4 = Fehlerkode (Ø = ok)
ØE8C        addq.l  #4,a7               ; Stackzeiger korrigieren
ØE8E F8ØE8E movem.l (a7)+,d2-d6/a2-a3   ; Register wiederherstellen
ØE92        rts
ØE94
ØE94 ;----- Motor einschalten
ØE94
ØE94 F8ØE94 move.l  a2,-(a7)            ; a2 retten
ØE96        movea.l 8(a7),a2            ; a2 -> Stackrahmen
ØE9A        btst.b  #7,1(a2)            ; Motorflag gesetzt?
ØEAØ        bne     F8ØEC4              ; Ja: fertig ->
ØEA4        bset.b  #7,1(a2)            ; Motorflag setzen
ØEAA        move.l  a2,-(a7)            ; Adresse des Stackrahmens übergeben
ØEAC        jsr     -12E(pc)            ; ---> F8ØD8Ø: Motor einschalten
ØEBØ        move.l  a2,-(a7)            ; Adresse des Stackrahmens übergeben
ØEB2        jsr     -1ØE(pc)            ; ---> F8ØDA6: Laufwerke abwählen
ØEB6        move.l  #7A12Ø,-(a7)        ; Wartezahl 5ØØØØØ: 1 s
ØEBC        jsr     -1ØA(pc)            ; ---> F8ØDB4: Warten
ØECØ        lea     C(a7),a7            ; Stackzeiger korrigieren
ØEC4 F8ØEC4 movea.l (a7)+,a2            ; a2 wiederherstellen
ØEC6        rts
ØEC8
ØEC8 ;----- Motor ausschalten
ØEC8
ØEC8 F8ØEC8 move.l  a2,-(a7)            ; a2 retten
ØECA        movea.l 8(a7),a2            ; a2 -> Stackrahmen
ØECE        btst.b  #7,1(a2)            ; Motorflag gesetzt?
```

```
ØED4          beq       F8ØEEC              ; Nein: fertig ->
ØED8          bclr.b    #7,1(a2)            ; Motorflag löschen
ØEDE          move.l    a2,-(a7)            ; Adresse des Stackrahmens übergeben
ØEEØ          jsr       -162(pc)            ; ---> F8ØD8Ø: Motor ausschalten
ØEE4          move.l    a2,-(a7)            ; Adresse des Stackrahmens übergeben
ØEE6          jsr       -142(pc)            ; ---> F8ØDA6: Laufwerke abwählen
ØEEA          addq.l    #8,a7               ; Stackzeiger korrigieren
ØEEC F8ØEEC   movea.l (a7)+,a2              ; a2 wiederherstellen
ØEEE          rts
ØEFØ
ØEFØ ;-----   Tabelle für Initialisierung des Stackrahmens
ØEFØ
ØEFØ F8ØEFØ   DC.B      Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
ØFØ4
ØFØ4 ;-----   8-Byte-Gruppe dekodieren
ØFØ4
ØFØ4 F8ØFØ4   move.l    d2,-(a7)            ; d2 retten
ØFØ6          move.l    8(a7),dØ            ; dØ -> 8-Byte-Gruppe im Puffer
ØFØA          move.l    #55555555,d2        ; d2 := Bitmaske Ø1Ø1Ø1Ø1...
ØF1Ø          movea.l dØ,aØ                 ; aØ -> 8-Byte-Gruppe
ØF12          addq.l    #4,aØ               ; aØ -> Gruppe der geraden Bits
ØF14          move.l    (aØ),d1             ; d1 := Gruppe der geraden Bits
ØF16          and.l     d2,d1               ; Zwischenbits löschen
ØF18          movea.l dØ,aØ                 ; aØ -> 8-Byte-Gruppe
ØF1A          move.l    (aØ),dØ             ; dØ := Gruppe der ungeraden Bits
ØF1C          and.l     d2,dØ               ; Zwischenbits löschen
ØF1E          lsl.l     #1,dØ               ; Ungerade Bits auf ungerade Stellen
ØF2Ø          or.l      dØ,d1               ; und mit den geraden Bits odern
ØF22          move.l    d1,dØ               ; dØ := d1 = dekodierte Gruppe
ØF24          move.l    (a7)+,d2            ; d2 wiederherstellen
ØF26          rts
ØF28
ØF28 ;-----   Block aus Puffer dekodieren und in Kickstart-RAM kopieren
ØF28
ØF28 F8ØF28   movem.l d2-d4/a2,-(a7)        ; Register retten
ØF2C          move.l    14(a7),dØ           ; dØ := Länge einer Gruppe = 512 Bytes
ØF3Ø          move.l    18(a7),d3           ; d3 := Zieladresse
ØF34          move.l    1C(a7),d1           ; d1 -> Anfang der Daten im Sektor
ØF38          move.l    #55555555,d2        ; d2 := Bismaske Ø1Ø1Ø1Ø1...
ØF3E          movea.l d1,a1                 ; a1 -> Anfang der Daten im Sektor
```

```
ØF4Ø         add.l    dØ,d1            ; d1 -> Anfang der 2. Datengruppe
ØF42         movea.l d1,a2             ; a2 := d1 -> Anfang der 2. Datengruppe
ØF44         move.l   dØ,d4            ; d4 := dØ = Blocklänge
ØF46         movea.l d3,aØ             ; aØ := d3 = Zieladresse
ØF48         bra.s    F8ØF5A           ; --->
ØF4A
ØF4A F8ØF4A  move.l   (a2)+,d1         ; d1 := Langwort aus 2. Gruppe (gerade)
ØF4C         and.l    d2,d1            ; Zwischenbits löschen
ØF4E         move.l   (a1)+,dØ         ; dØ := Langwort aus 1. Gruppe (unger.)
ØF5Ø         and.l    d2,dØ            ; Zwischenbits löschen
ØF52         lsl.l    #1,dØ            ; Ungerade Bits 1 nach links
ØF54         or.l     dØ,d1            ; und mit geraden Bits odern
ØF56         move.l   d1,(aØ)+         ; Ergebnis an Zieladresse schreiben
ØF58         subq.l   #4,d4            ; Blocklänge um Langwort vermindern
ØF5A F8ØF5A  tst.l    d4               ; Block fertig?
ØF5C         bhi.s    F8ØF4A           ; Nein: weitermachen ->
ØF5E         movem.l (a7)+,d2-d4/a2    ; Register wiederherstellen
ØF62         rts
ØF64
ØF64 ;-----  Ende der Spurlücke suchen
ØF64
ØF64 F8ØF64  move.l   d2,-(a7)         ; d2 retten
ØF66         move.l   8(a7),dØ         ; dØ -> Anfang der Lücke
ØF6A         move.l   C(a7),d1         ; d1 := 3428 = max. Länge der Lücke
ØF6E         move.l   dØ,d2            ; d2 := dØ -> Anfang der Lücke
ØF7Ø         movea.l dØ,aØ             ; aØ := dØ -> Anfang der Lücke
ØF72         adda.l   d1,aØ            ; aØ -> Anfang der Lücke + 3428
ØF74         move.l   aØ,dØ            ; dØ -> Anfang der Lücke + 3428
ØF76 F8ØF76  cmp.l    dØ,d2            ; Max. Länge der Lücke überschritten?
ØF78         bcc.s    F8ØFAØ           ; Ja: Fehler ->
ØF7A         movea.l d2,aØ             ; aØ := dØ = Suchzeiger
ØF7C         cmpi.w   #4489,(aØ)       ; Sync-Wort gefunden?
ØF8Ø         bne.s    F8ØF9C           ; Nein: weitersuchen ->
ØF82         movea.l d2,aØ             ; aØ := d2 = Suchzeiger
ØF84         addq.l   #2,aØ            ; um 2 erhöhen
ØF86         cmpi.w   #4489,(aØ)       ; Folgt zweites Sync-Wort?
ØF8A         bne.s    F8ØF94           ; Nein: Erstes Sync-Wort verloren ->
ØF8C         movea.l d2,aØ             ; aØ := d2 -> 1. Sync-Wort
ØF8E         subq.l   #4,aØ            ; aØ um 4 Bytes zurücksetzen
ØF9Ø         move.l   aØ,dØ            ; dØ := aØ -> Sektoranfang nach Lücke
```

```
ØF92          bra.s    F8ØFA2              ; --->
ØF94
ØF94 F8ØF94   movea.l d2,aØ                ; aØ := d2 -> 2. Sync-Wort
ØF96          subq.l  #6,aØ                ; aØ um 6 Bytes zurücksetzen
ØF98          move.l  aØ,dØ                ; dØ := aØ -> Sektoranfang nach Lücke
ØF9A          bra.s   F8ØFA2               ; --->
ØF9C
ØF9C F8ØF9C   addq.l  #2,d2                ; Suchzeiger auf nächstes Wort setzen
ØF9E          bra.s   F8ØF76               ; --->
ØFAØ
ØFAØ F8ØFAØ   moveq   #Ø,dØ                ; Flag: Kein Sektoranfang
ØFA2 F8ØFA2   move.l  (a7)+,d2             ; d2 wiederherstellen
ØFA4          rts
ØFA6
ØFA6 ;-----   Gelesene Spur analysieren und prüfen, Lücke suchen
ØFA6
ØFA6 F8ØFA6   link    a6,#-4               ; Platz für Sektor-Header reservieren
ØFAA          movem.l d2-d6/a2-a3,-(a7)    ; Register retten
ØFAE          movea.l 8(a6),a2             ; a2 -> Stackrahmen
ØFB2          moveq   #Ø,d4                ; d4 := Ø
ØFB4          movea.l 6(a2),aØ             ; aØ -> Lesepuffer
ØFB8          addq.l  #6,aØ                ; aØ um 6 erhöhen
ØFBA          move.l  aØ,dØ                ; dØ := aØ
ØFBC          movea.l dØ,aØ                ; aØ := dØ
ØFBE          cmpi.w  #4489,(aØ)           ; Steht Sync-Wort am Anfan
ØFC2          beq.s   F8ØFCA               ; Ja: ->
ØFC4          moveq   #16,d4               ; d4 := 22 = Fehlerkode
ØFC6          bra     F81ØC4               ; ---> Ausgang
ØFCA
ØFCA F8ØFCA   movea.l dØ,aØ                ; aØ -> Leseanfang
ØFCC          addq.l  #2,aØ                ; aØ um 2 erhöhen
ØFCE          cmpi.w  #4489,(aØ)           ; Folgt das zweite Sync-Wort?
ØFD2          bne.s   F8ØFDC               ; Nein: ->
ØFD4          movea.l dØ,aØ                ; aØ -> Leseanfang
ØFD6          subq.l  #4,aØ                ; aØ um 4 erniedrigen
ØFD8          movea.l aØ,a3                ; a3 -> 4 Bytes vor Leseanfang
ØFDA          bra.s   F8ØFE2               ; --->
ØFDC
ØFDC F8ØFDC   movea.l dØ,aØ                ; aØ -> Leseanfang
ØFDE          subq.l  #6,aØ                ; aØ um 6 erniedrigen
```

```
0FE0          movea.l a0,a3          ; a3 -> 6 Bytes vor Leseanfang
0FE2 F80FE2   move.w  #AAAA,(a3)     ; 4 mal MFM-Kode $AA eintragen
0FE6          move.w  #AAAA,2(a3)
0FEC          move.w  #4489,4(a3)    ; Sync-Wort eintragen
0FF2          move.l  a3,A(a2)       ; Anfang in Stackrahmen eintragen
0FF6          pea     8(a3)          ; Anfang der Header-Info übergeben
0FFA          jsr     F80F04         ; ---> Header-Info dekodieren
1000          move.l  d0,-4(a6)      ; Header-Info abspeichern
1004          pea     -4(a6)         ; Adresse der dekodierten Header-Info
1008          move.l  a3,-(a7)       ; Anfang der Spur im Puffer
100A          move.l  a2,-(a7)       ; Adresse des Stackrahmens
100C          jsr     F81114         ; ---> Header prüfen
1012          move.l  d0,d4          ; d4 := d0 = Fehlerkode
1014          lea     10(a7),a7      ; Stackzeiger korrigieren
1018          bne     F810C4         ; Fehler ist aufgetreten: ->
101C          move.b  -1(a6),3(a2)   ; Sektor-Offset bis Lücke
1022          move.b  -2(a6),2(a2)   ; Sektor-Nummer
1028          cmpi.b  #B,3(a2)       ; Sektor-Offset > 10?
102E          bcc.s   F8105E         ; Ja: Lücke am Ende des Puffers ->
1030          pea     D64            ; 3428 (maximale Länge der Lücke)
1034          moveq   #0,d2          ; d2 := 0
1036          move.b  3(a2),d2       ; d2 := Offset zur Lücke (Blöcke)
103A          mulu    #440,d2        ; mal 1088 (Blocklänge im Puffer)
103E          movea.l d2,a0          ; a0 := Offset zur Lücke in Bytes
1040          adda.l  a3,a0          ; + Leseanfang = Anfang der Lücke
1042          pea     (a0)           ; auf Stack übergeben
1044          jsr     -E2(pc)        ; ---> F80F64: Ende der Lücke suchen
1048          movea.l d0,a3          ; a3 := d0 -> 1. Sektor nach der Lücke
104A          move.l  a3,E(a2)       ; abspeichern
104E          exg     d6,a3          ; 'tst' geht nicht mit Adreßregister
1050          tst.l   d6             ; Sync-Wort nach der Lücke gefunden?
1052          exg     d6,a3          ; a3 wiederherstellen
1054          addq.l  #8,a7          ; Stackzeiger korrigieren
1056          bne.s   F81062         ; Sync-Wort gefunden: ->
1058          moveq   #17,d4         ; d4 := 23 = Fehlerkode
105A          bra     F810C4         ; ---> Ausgang
105E
105E F8105E   clr.l   E(a2)          ; weil Lücke am Ende des Puffers
1062 F81062   moveq   #0,d5          ; Anfangswert der Sektornummer
1064 F81064   move.l  d5,-(a7)       ; Sektornummer übergeben
```

```
1066        move.l  a2,-(a7)          ; Adresse des Stackrahmens übergeben
1068        jsr     F810CE            ; ---> Anfang des Sektors ermitteln
106E        movea.l d0,a3             ; a3 := d0 = Anfang des Sektors
1070        pea     8(a3)             ; Adresse der Header-Info übergeben
1074        jsr     F80F04            ; ---> Header-Info dekodieren
107A        move.l  d0,-4(a6)         ; und abspeichern
107E        pea     -4(a6)            ; Adresse der Header-Info übergeben
1082        move.l  a3,-(a7)          ; Anfangsadresse des Sektors übergeben
1084        move.l  a2,-(a7)          ; Adresse des Stackrahmens übergeben
1086        jsr     F81114            ; ---> Header prüfen
108C        move.l  d0,d4             ; d4 := d0 = Fehlerkode
108E        lea     18(a7),a7         ; Stackzeiger korrigieren
1092        bne     F810C4            ; Header fehlerhaft: ->
1096        pea     38(a3)            ; Adresse der Block-Prüfsumme übergeben
109A        jsr     F80F04            ; ---> Prüfsumme dekodieren
10A0        move.l  d0,d3             ; d3 := d0 = dekodierte Prüfsumme
10A2        pea     400               ; 1024 (Datenbytes im Puffer) übergeben
10A6        pea     40(a3)            ; Anfang der Daten im Puffer übergeben
10AA        jsr     F81184            ; ---> Prüfsumme für Daten berechnen
10B0        cmp.l   d0,d3             ; Prüfsumme berechnet wie gelesen?
10B2        lea     C(a7),a7          ; Stackzeiger korrigieren
10B6        beq.s   F810BC            ; Prüfsumme ok: ->
10B8        moveq   #18,d4            ; d4 := 24 = Fehlerkode
10BA        bra.s   F810C4            ; ---> Ausgang
10BC
10BC F810BC addq.l  #1,d5             ; Sektornummer inkrementieren
10BE        moveq   #B,d0             ; d0 := 11 = höchste Sektornummer+1
10C0        cmp.l   d5,d0             ; Alle Sektoren bearbeitet?
10C2        bgt.s   F81064            ; Nein: weitermachen ->
10C4 F810C4 move.l  d4,d0             ; d0 := d4 = Fehlerkode
10C6        movem.l (a7)+,d2-d6/a2-a3 ; Register wiederherstellen
10CA        unlk    a6                ; Platz für Header-Info freigeben
10CC        rts
10CE
10CE ;----- Adresse eines Sektors aus seiner Nummer ermitteln
10CE
10CE F810CE movem.l d2-d5,-(a7)       ; Register retten
10D2        movea.l 14(a7),a0         ; a0 -> Stackrahmen
10D6        move.l  18(a7),d2         ; d2 := Nummer des gesuchten Sektors
10DA        moveq   #0,d3             ; d3 := 0
```

```
10DC        move.b  2(a0),d3        ; d3 := Nummer des 1. Sektors im Puffer
10E0        moveq   #0,d4           ; d4 := 0
10E2        move.b  3(a0),d4        ; d4 := Zahl der Sektoren vor der Lücke
10E6        move.l  A(a0),d5        ; d5 := Anfangsadr der gelesenen Spur
10EA F810EA cmp.l   d3,d2           ; Sektornummer gefunden?
10EC        bne.s   F810F2          ; Nein: ->
10EE        move.l  d5,d0           ; d0 := d5 = Adresse des Sektors
10F0        bra.s   F8110E          ; ---> Ausgang
10F2
10F2 F810F2 addi.l  #440,d5         ; Adreßzeiger um Sektorlänge erhöhen
10F8        subq.l  #1,d4           ; Sektorzahl vor der Lücke erniedrigen
10FA        bne.s   F81100          ; Lücke noch nicht erreicht: ->
10FC        move.l  E(a0),d5        ; d5 := Adresse 1. Sektor nach der Lücke
1100 F81100 addq.l  #1,d3           ; d3 := nächste Sektornummer
1102        move.l  d3,d1           ; d1 := d3 = aktuelle Sektornummer
1104        moveq   #B,d0           ; d0 := 11 = höchste Sektornummer+1
1106        cmp.l   d1,d0           ; Aktuelle Sektornummer > 10?
1108        bgt.s   F810EA          ; Nein: ->
110A        moveq   #0,d3           ; Sonst durch Sektornummer 0 ersetzen
110C        bra.s   F810EA          ; ---> weitersuchen
110E
110E F8110E movem.l (a7)+,d2-d5     ; Register wiederherstellen
1112        rts
1114
1114 ;----- Sektor-Header prüfen
1114
1114 F81114 movem.l d2/a2-a4,-(a7)  ; Register retten
1118        movea.l 14(a7),a4       ; a4 -> Stackrahmen
111C        movea.l 18(a7),a3       ; a3 -> Anfang des Headers
1120        movea.l 1C(a7),a2       ; a2 -> dekodierte Header-Info
1124        cmpi.w  #4489,6(a3)     ; Sync-Wort an der richtigen Stelle?
112A        bne     F81150          ; Nein: Fehler ->
112E        pea     30(a3)          ; Adresse der Header-Prüfsumme übergeben
1132        jsr     F80F04          ; ---> Prüfsumme dekodieren
1138        move.l  d0,d2           ; d2 := d0 = Prüfsumme
113A        pea     28              ; 40 = Anzahl der Summanden übergeben
113E        pea     8(a3)           ; Anfang für Prüfsumme übergeben
1142        jsr     F81184          ; ---> Prüfsumme ermitteln
1148        cmp.l   d0,d2           ; Prüfsumme berechnet wie gelesen?
114A        lea     C(a7),a7        ; Stackzeiger korrigieren
```

```
114E         beq.s    F81154            ; Prüfsumme ok: ->
1150 F81150  moveq    #15,d0            ; d0 := 21 = Fehlerkode
1152         bra.s    F8117E            ; ---> Ausgang
1154
1154 F81154  cmpi.b   #FF,(a2)          ; Formatbyte richtig?
1158         bne.s    F81150            ; Nein: Fehler ->
115A         moveq    #0,d0             ; d0 := 0
115C         move.b   1(a2),d0          ; d0 := Nummer der gelesenen Spur
1160         cmp.w    4(a4),d0          ; = Spurnummer im Header?
1164         bne.s    F81150            ; Nein: Fehler ->
1166         cmpi.b   #B,2(a2)          ; Sektornummer im Header < 11?
116C         bcc.s    F81150            ; Nein: Fehler ->
116E         tst.b    3(a2)             ; Offset zur Lücke
1172         beq.s    F81150            ; = 0: Fehler ->
1174         cmpi.b   #B,3(a2)          ; > 11?
117A         bhi.s    F81150            ; Ja: Fehler ->
117C         moveq    #0,d0             ; ok-Flag
117E F8117E  movem.l  (a7)+,d2/a2-a4    ; Register wiederherstellen
1182         rts
1184
1184 ;-----  Prüfsumme ermitteln
1184
1184 F81184  move.l   d2,-(a7)          ; d2 retten
1186         movea.l  8(a7),a0          ; a0 -> Anfang für Prüfsummenermittlung
118A         move.l   C(a7),d0          ; d0 := Bytezahl für Prüfsumme
118E         moveq    #0,d1             ; d1 := 0 = Ergebnisregister
1190         asr.l    #2,d0             ; d0 := Zahl der Langworte für Prüfsumme
1192 F81192  tst.l    d0                ; Langwortzähler abgelaufen?
1194         beq.s    F8119E            ; Ja: fertig ->
1196         subq.l   #1,d0             ; Zähler dekrementieren
1198         move.l   (a0)+,d2          ; d2 := nächstes Langwort
119A         eor.l    d2,d1             ; Mit Ergebnisregister exklusiv odern
119C         bra.s    F81192            ; ---> wiederholen
119E
119E F8119E  andi.l   #55555555,d1      ; Ungerade Bits in d1 löschen
11A4         move.l   d1,d0             ; d0 := d1 = Ergebnis
11A6         move.l   (a7)+,d2          ; d2 wiederherstellen
11A8         rts
11AA
11AA         DC.W     0
```

```
11AC
11AC ;-----  Graphikausgabe vorbereiten
11AC
11AC F811AC  movem.l d2-d3/a2-a4,-(a7)  ; Register retten
11B0         move.l  18(a7),d3          ; d3 -> Bereich für Copperliste
11B4         movea.l #6000,a4           ; a4 -> BitPlane 2
11BA         movea.l #4000,a3           ; a3 -> BitPlane 1
11C0         movea.l #DFF000,a2         ; a2 := ChipBase
11C6         moveq   #0,d2              ; d2 := 0 = BitPlane-Zähler
11C8 F811C8  move.w  d2,d0              ; d0 := d2 = Nummer der BitPlane
11CA         add.w   d0,d0              ; verdoppeln
11CC         lea     110(a2),a0         ; a0 -> BPL1DAT: BitPlane Datenregister
11D0         clr.w   0(a0,d0.w)         ; Register für BitPlane (d0) löschen
11D4         move.w  d2,d0              ; d0 := d2 = Nummer der BitPlane
11D6         asl.w   #2,d0              ; mal 4
11D8         lea     E0(a2),a0          ; a0 := BPL1PTH: BitPlane Zeiger
11DC         clr.l   0(a0,d0.w)         ; Zeiger für BitPlane (d0) löschen
11E0         addq.l  #1,d2              ; Nummer der BitPlane erhöhen
11E2         moveq   #6,d0              ; d0 := 6 = höchste BitPlane-Nummer+1
11E4         cmp.l   d2,d0              ; Alle BitPlanes bearbeitet?
11E6         bgt.s   F811C8             ; Nein: weitermachen ->
11E8         moveq   #0,d2              ; d2 := 0 = Sprite-Zähler
11EA F811EA  move.w  d2,d0              ; d0 := d2 = Sprite-Nummer
11EC         asl.w   #3,d0              ; mal 8
11EE         lea     140(a2),a0         ; a0 := SPR0POS: Sprite x-Anfangswert
11F2         move.w  #FE00,0(a0,d0.w)   ; hinter das Zeilenende legen
11F8         move.w  d2,d0              ; d0 := d2 = Sprite-Nummer
11FA         asl.w   #3,d0              ; mal 8
11FC         lea     140(a2),a0         ; a0 := SPR0POS
1200         move.w  #FF00,2(a0,d0.w)   ; x-Endwert noch dahinter
1206         addq.l  #1,d2              ; Sprite-Nummer erhöhen
1208         moveq   #8,d0              ; d0 := 8 = höchste Sprite-Nummer + 1
120A         cmp.l   d2,d0              ; Alle Sprites bearbeitet?
120C         bgt.s   F811EA             ; Nein: weitermachen ->
120E         clr.l   -(a7)              ; Löschkode 0 für BitPlanes
1210         pea     1F40               ; 8000 = Länge der BitPlanes in Bytes
1214         pea     (a3)               ; Adresse der BitPlane 1
1216         jsr     F81540             ; ---> BitPlane löschen
121C         clr.l   -(a7)              ; Löschkode 0 für BitPlanes
121E         pea     1F40               ; 8000 = Länge der BitPlanes in Bytes
```

```
1222            pea     (a4)            ; Adresse der BitPlane 2
1224            jsr     F81540          ; ---> BitPlane löschen
122A            movea.l d3,a1           ; a1 := d3 -> Anfang der Copperliste
122C            lea     96(a2),a0       ; a0 -> DMACON
1230            move.w  a0,d2           ; d2 -> DMACON
1232            andi.w  #1FE,d2         ; d2 := Zielregister für Copper-MOVE
1236            move.w  d2,(a1)+        ; Wort in Copperliste schreiben
1238            move.w  #80,(a1)+       ; Daten für MOVE: COPEN löschen
123C            move.w  #FFFF,(a1)+     ; Copper-WAIT bis Bildwechsel
1240            move.w  #FFFE,(a1)+     ; in Copperliste schreiben
1244            move.w  #80,96(a2)      ; DMACON: Copper DMA Enable löschen
124A            move.l  d3,80(a2)       ; COP1LCH := Adresse der Copperliste
124E            move.w  #1,88(a2)       ; COPJMP1 := 1 startet Copper
1254            move.w  #8080,96(a2)    ; DMACON: Copper DMA Enable setzen
125A            moveq   #0,d2           ; d2 := 0 = Zeitzähler
125C            lea     18(a7),a7       ; Stackzeiger korrigieren
1260            bra.s   F81272          ; --->
1262
1262 F81262     cmpi.l  #2710,d2        ; d2 > 10000?
1268            ble.s   F81270          ; Nein: ->
126A            jsr     F81342          ; ---> Fehler-Graphik ausgeben
1270 F81270     addq.l  #1,d2           ; Zeitzähler erhöhen
1272 F81272     btst.b  #7,3(a2)        ; DMACONR: COPEN noch gesetzt?
1278            bne.s   F81262          ; Ja: warten ->
127A            movea.l d3,a1           ; a1 := d3 -> Copperliste
127C            lea     E0(a2),a0       ; a0 -> BPL1PTH
1280            move.w  a0,d2           ; d2 := a0 -> BPL1PTH
1282            andi.w  #1FE,d2         ; d2 := Zielregister für Copper-MOVE
1286            move.w  d2,(a1)+        ; in Copperliste schreiben
1288            move.l  a3,d0           ; d0 := a3 -> BitPlane 1
128A            moveq   #10,d1          ; d1 := 16
128C            asr.l   d1,d0           ; d0.w := BitPlane 1 Adresse,H
128E            move.w  d0,(a1)+        ; in Copperliste schreiben
1290            lea     E0(a2),a0       ; a0 -> BPL1PTH
1294            move.w  a0,d2           ; d2 := a0 -> BPL1PTH
1296            addq.w  #2,d2           ; d2 -> BPL1PTL
1298            andi.w  #1FE,d2         ; d2 := Zielregister für Copper-MOVE
129C            move.w  d2,(a1)+        ; in Copperliste schreiben
129E            movea.l a3,a0           ; a0 := a3 -> BitPlane 1
12A0            move.w  a0,d2           ; d2 := a0 -> BitPlane 1
```

```
12A2    andi.w   #FFFF,d2           ; d2.w := BitPlane 1 Adresse,L
12A6    move.w   d2,(a1)+           ; in Copperliste schreiben
12A8    lea      E4(a2),a0          ; a0 -> BPL2PTH
12AC    move.w   a0,d2              ; d2 := a0 -> BPL2PTH
12AE    andi.w   #1FE,d2            ; d2 := Zielregister für Copper-MOVE
12B2    move.w   d2,(a1)+           ; in Copperliste schreiben
12B4    move.l   a4,d0              ; d0 := a4 -> BitPlane 2
12B6    moveq    #10,d1             ; d1 := 16
12B8    asr.l    d1,d0              ; d0.w := BitPlane 2 Adresse,H
12BA    move.w   d0,(a1)+           ; in Copperliste schreiben
12BC    lea      E4(a2),a0          ; a0 -> BPL2PTH
12C0    move.w   a0,d2              ; d2 := a0 -> BPL2PTH
12C2    addq.w   #2,d2              ; d2 -> BPL2PTL
12C4    andi.w   #1FE,d2            ; d2 := Zielregister für Copper-MOVE
12C8    move.w   d2,(a1)+           ; in Copperliste schreiben
12CA    movea.l  a4,a0              ; a0 := a4 -> BitPlane 2
12CC    move.w   a0,d2              ; d2 := a0 -> BitPlane 2
12CE    andi.w   #FFFF,d2           ; d2.w := BitPlane 2 Adresse,L
12D2    move.w   d2,(a1)+           ; in Copperliste schreiben
12D4    move.w   #FFFF,(a1)+        ; Copper-WAIT bis Bildwechsel
12D8    move.w   #FFFE,(a1)+        ; in Copperliste schreiben
12DC    move.l   a3,E0(a2)          ; BPL1PT := a3
12E0    move.l   a4,E4(a2)          ; BPL2PT := a4
12E4    move.l   d3,80(a2)          ; COP1LC := d3
12E8    move.w   #FFF,180(a2)       ; COLOR00 := Weiß
12EE    move.w   #0F0,182(a2)       ; COLOR01 := Grün
12F4    move.w   #F00,184(a2)       ; COLOR02 := Rot
12FA    move.w   #00F,186(a2)       ; COLOR03 := Blau
1300    clr.w    108(a2)            ; BPL1MOD := 0
1304    clr.w    10A(a2)            ; BPL2MOD := 0
1308    move.w   #2300,100(a2)      ; BPLCON0 := BPU1, COLOR, GAUD
130E    clr.w    102(a2)            ; BPLCON1 := 0
1312    move.w   #24,104(a2)        ; BPLCON2 := PF2P2, PF1P2
1318    move.w   #2C81,8E(a2)       ; DIWSTRT: y =  44, x = 129
131E    move.w   #F4C1,90(a2)       ; DIWSTOP: y = 244, x = 193
1324    move.w   #38,92(a2)         ; DDFSTRT: normal
132A    move.w   #D0,94(a2)         ; DDFSTOP: normal
1330    move.w   #1,88(a2)          ; COPJMP1 := 1: Copper starten
1336    move.w   #8080,96(a2)       ; DMACON: Copper DMA Enable setzen
133C    movem.l  (a7)+,d2-d3/a2-a4  ; Register wiederherstellen
```

```
1340        rts
1342
1342 ;----- Fehlergraphik ausgeben
1342
1342 F81342 movem.l d2/a2,-(a7)        ; Register retten
1346        movea.l #DFF000,a2         ; a2 := ChipBase
134C        jsr     F813D6             ; ---> Warten auf Bildwechsel
1352        move.w  #F80,180(a2)       ; COLOR00 := Rot-Orange
1358        clr.w   184(a2)            ; COLOR02 := Schwarz
135C        clr.w   186(a2)            ; COLOR03 := Schwarz
1360        move.w  #8180,96(a2)       ; DMACON: BPLEN und COPEN setzen
1366        pea     F8195C             ; Adresse der BitPlane-Daten übergeben
136C        jsr     F81480             ; ---> BitPlanes erzeugen
1372        addq.l  #4,a7              ; Stackzeiger korrigieren
1374 F81374 moveq   #0,d2              ; d2 := 0 = Zähler für Blinkdauer
1376 F81376 jsr     F813D6             ; ---> Warten auf Bildwechsel
137C        addq.l  #1,d2              ; Zähler d2 inkrementieren
137E        moveq   #1E,d0             ; d0 := 30 = Grenzwert
1380        cmp.l   d2,d0              ; Grenzwert erreicht?
1382        bgt.s   F81376             ; Nein: warten ->
1384        move.w  #F80,184(a2)       ; COLOR02 := Rot-Orange
138A        moveq   #0,d2              ; d2 := 0 = Zähler für Blinkdauer
138C F8138C jsr     F813D6             ; ---> Warten auf Bildwechsel
1392        addq.l  #1,d2              ; Zähler d2 inkrementieren
1394        moveq   #1E,d0             ; d0 := 30 = Grenzwert
1396        cmp.l   d2,d0              ; Grenzwert erreicht?
1398        bgt.s   F8138C             ; Nein: warten ->
139A        clr.w   184(a2)            ; COLOR02 := Schwarz
139E        bra.s   F81374             ; Blinken ohne Ende ->
13A0
13A0 ;----- Farben setzen, BitPlanes erzeugen
13A0
13A0 F813A0 move.l  a2,-(a7)           ; a2 retten
13A2        movea.l #DFF000,a2         ; a2 := ChipBase
13A8        jsr     F813D6             ; ---> Warten auf Bildwechsel
13AE        move.w  #FFF,180(a2)       ; COLOR00 := Weiß
13B4        clr.w   182(a2)            ; COLOR01 := Schwarz
13B8        move.w  #77C,184(a2)       ; COLOR02 := Blau
13BE        move.w  #CCC,186(a2)       ; COLOR03 := Hellgrau
13C4        pea     F81A00             ; Adresse der BitPlane-Daten übergeben
```

```
13CA        jsr     F81480          ; ---> BitPlanes erzeugen
13D0        addq.l  #4,a7           ; Stackzeiger korrigieren
13D2        movea.l (a7)+,a2        ; a2 wiederherstellen
13D4        rts
13D6
13D6 ;----- Warten auf Bildwechsel
13D6
13D6 F813D6 movea.l #DFF000,a0      ; a0 := ChipBase
13DC        move.w  #20,9C(a0)      ; INTREQ: VertB-Bit löschen
13E2 F813E2 btst.b  #5,1F(a0)       ; INTREQR: VertB-Bit gesetzt?
13E8        beq.s   F813E2          ; Nein: warten ->
13EA        rts
13EC
13EC ;----- Muster in BitPlane-Rechteck eintragen
13EC
13EC F813EC movem.l d2-d7/a2-a3,-(a7) ; Register retten
13F0        movea.l 24(a7),a0       ; a0 -> Musterdaten in Tabelle
13F4        move.l  28(a7),d3       ; d3 := Worte pro Zeile
13F8        move.l  2C(a7),d1       ; d1 -> BitPlane
13FC        moveq   #0,d5           ; d5 := 0
13FE        move.w  (a0)+,d5        ; d5 := Zeilen im Rechteck
1400        moveq   #0,d2           ; d2 := 0
1402        move.w  (a0)+,d2        ; d2 := Pixel-Offset in BitPlane
1404        move.l  d2,d6           ; d6 := d2 = Pixeloffset
1406        moveq   #F,d0           ; d0 := 15
1408        and.l   d0,d6           ; d6 := Pixeloffset modulo 16
140A        move.l  d2,d0           ; d0 := Pixeloffset
140C        asr.l   #4,d0           ; d0 := Pixeloffset/16 = Wortoffset
140E        add.l   d0,d0           ; d0 := 2*Wortoffset = Byteoffset
1410        movea.l d0,a1           ; a1 := Byteoffset in BitPlane
1412        adda.l  d1,a1           ; a1 := Anfangsadresse des Musters
1414        movea.l a1,a2           ; a2 := a1 = Anfangsadresse des Musters
1416        tst.l   d6              ; Pixeloffset modulo 16 = 0?
1418        bne.s   F81440          ; Nein: ->
141A F8141A move.l  d5,d0           ; d0 := d5 = Zeilenzähler
141C        subq.l  #1,d5           ; Zähler um 1 vermindern
141E        move.l  d0,d0           ; Statusflags setzen
1420        beq     F81478          ; Zeilenzähler = 0: fertig ->
1424        movea.l d3,a1           ; a1 := d3 = Wortzähler
1426        bra.s   F8142C          ; --->
```

```
1428
1428 F81428  move.w  (a0)+,d1          ; d1 := nächstes Tabellenwort
142A         or.w    d1,(a2)+          ; Tabellenwort in BitPlane odern
142C F8142C  move.l  a1,d0             ; d0 := a1 = Wortzähler
142E         subq.l  #1,a1             ; um 1 vermindern
1430         move.l  d0,d0             ; Statusflags setzen
1432         bne.s   F81428            ; Wortzähler noch nicht Null: ->
1434         moveq   #14,d7            ; d7 := 20 Worte pro 320 Pixel
1436         movea.l d7,a1             ; a1 := d7 = 20
1438         suba.l  d3,a1             ; - Zahl der Worte/Musterzeile
143A         adda.l  a1,a1             ; a1 := Bytezahl bis nächste Musterzeile
143C         adda.l  a1,a2             ; a2 := Anfangsadr. nächste Musterzeile
143E         bra.s   F8141A            ; ---> wiederholen
1440
1440 F81440  moveq   #10,d4            ; d4 := 16
1442         sub.l   d6,d4             ; d4 := 16 - Pixeloffset modulo 16
1444 F81444  move.l  d5,d0             ; d0 := d5 = Zeilenzähler
1446         subq.l  #1,d5             ; Zähler um 1 vermindern
1448         move.l  d0,d0             ; Statusflags setzen
144A         beq.s   F81478            ; Zeilenzähler = 0: fertig ->
144C         clr.w   d2                ; d2 := 0
144E         movea.l d3,a1             ; a1 := d3 = Wortzähler
1450         bra.s   F81464            ; --->
1452
1452 F81452  movea.w (a0)+,a3          ; a3 := nächstes Wort aus Tabelle
1454         move.w  a3,d1             ; d1 := a3 = Wort aus Tabelle
1456         move.b  d6,d0             ; d0 := d6 = Pixeloffset mod 16
1458         lsr.w   d0,d1             ; d1 um d0 Bits nach rechts schieben
145A         move.b  d4,d0             ; d0 := d4 = 16-Pixeloffset mod 16
145C         lsl.w   d0,d2             ; d2 um d0 Bits nach links schieben
145E         or.w    d2,d1             ; und in d1 einodern
1460         or.w    d1,(a2)+          ; Ergebnis in BitPlane einodern
1462         move.w  a3,d2             ; d2 := Tabellenwort
1464 F81464  move.l  a1,d0             ; d0 := a1 = Wortzähler
1466         subq.l  #1,a1             ; a1 um 1 vermindern
1468         move.l  d0,d0             ; Statusflags setzen
146A         bne.s   F81452            ; Wortzähler nicht Null: ->
146C         moveq   #14,d7            ; d7 := 20 Worte pro 320 Pixel
146E         movea.l d7,a1             ; a1 := d7 = 20
1470         suba.l  d3,a1             ; - Zahl der Worte/Musterzeile
```

```
1472         adda.l  a1,a1               ; mal 2 ergibt Zahl der Bytes
1474         adda.l  a1,a2               ; a2 -> Anfang nächste Musterzeile
1476         bra.s   F81444              ; ---> wiederholen
1478
1478 F81478  move.l  a0,d0               ; d0 := a0 = Zeiger in Datentabelle
147A         movem.l (a7)+,d2-d7/a2-a3   ; Register wiederherstellen
147E         rts
1480
1480 ;-----  BitPlanes erzeugen
1480
1480 F81480  movem.l d2-d6/a2,-(a7)      ; Register retten
1484         movea.l 1C(a7),a2           ; a2 -> Anfang der BitPlane-Daten
1488         addq.l  #2,a2               ; Erstes Wort $FFFF überlesen
148A         move.w  (a2)+,d6            ; d6 := 1. Datenwort
148C         ext.l   d6                  ; auf Langwort erweitern
148E         move.w  (a2)+,d4            ; d4 := 2. Datenwort
1490         ext.l   d4                  ; auf Langwort erweitern
1492         move.w  (a2)+,d5            ; d5 := 3. Datenwort
1494         ext.l   d5                  ; auf Langwort erweitern
1496 F81496  move.w  (a2)+,d3            ; d3 := nächstes Datenwort
1498         ext.l   d3                  ; auf Langwort erweitern
149A         move.w  (a2)+,d2            ; d2 := nächstes Datenwort
149C         ext.l   d2                  ; auf Langwort erweitern
149E         moveq   #FF,d0              ; d0 := -1
14A0         cmp.l   d3,d0               ; Datenwort -1 gelesen?
14A2         bne.s   F814B8              ; Nein: ->
14A4         moveq   #FF,d0              ; d0 := -1
14A6         cmp.l   d2,d0               ; Nächstes Datenwort auch -1?
14A8         beq     F8153A              ; Ja: fertig ->
14AC         move.l  d2,d6               ; d6 := d2 = Farbinformation
14AE         move.w  (a2)+,d4            ; d4 := nächstes Datenwort = x1
14B0         ext.l   d4                  ; auf Langwort erweitern
14B2         move.w  (a2)+,d5            ; d5 := nächstes Datenwort = y1
14B4         ext.l   d5                  ; auf Langwort erweitern
14B6         bra.s   F81496              ; ---> weiterlesen
14B8
14B8 F814B8  moveq   #FE,d0              ; d0 := -2
14BA         cmp.l   d3,d0               ; Datenwort -2 gelesen?
14BC         bne.s   F814E4              ; Nein: ->
14BE         move.w  (a2)+,d4            ; d4 := nächstes Datenwort = x
```

```
14C0        ext.l    d4            ; auf Langwort erweitern
14C2        move.w   (a2)+,d5      ; d5 := nächstes Datenwort = y
14C4        ext.l    d5            ; auf Langwort erweitern
14C6        move.l   d6,-(a7)      ; Farbinfo der Berandung
14C8        move.l   d2,-(a7)      ; Farbinfo des Untergrunds
14CA        move.l   d5,d2         ; d2 := d5 = y
14CC        asl.l    #2,d2         ; d2 := 4*y
14CE        move.l   d2,d3         ; d3 := 4*y
14D0        asl.l    #2,d2         ; d2 := 16*y
14D2        add.l    d3,d2         ; d2 := 20*y
14D4        move.l   d2,-(a7)      ; 20*y
14D6        move.l   d4,-(a7)      ; x
14D8        jsr      F817BC        ; ---> Fläche mit Randfarbe füllen
14DE        lea      10(a7),a7     ; Stackzeiger korrigieren
14E2        bra.s    F81496        ; ---> weiterlesen
14E4
14E4 F814E4 moveq    #FD,d0        ; d0 := -3
14E6        cmp.l    d3,d0         ; Datenwort -3 gelesen?
14E8        bne.s    F81500        ; Nein: ->
14EA        pea      4000          ; Anfangsadresse der BitPlane 1
14F0        move.l   d2,-(a7)      ; Zahl der Worte pro Musterzeile
14F2        move.l   a2,-(a7)      ; Zeiger in Datentabelle
14F4        jsr      -10A(pc)      ; ---> F813EC: Muster in BitPlane odern
14F8        movea.l  d0,a2         ; a2 := d0 = Zeiger in Datentabelle
14FA        lea      C(a7),a7      ; Stackzeiger korrigieren
14FE        bra.s    F81496        ; ---> weiterlesen
1500
1500 F81500 moveq    #FC,d0        ; d0 := -4
1502        cmp.l    d3,d0         ; Datenwort -4 gelesen?
1504        bne.s    F8151E        ; Nein: ->
1506        pea      6000          ; Anfangsadresse der BitPlane 2
150C        move.l   d2,-(a7)      ; Zahl der Worte pro Musterzeile
150E        move.l   a2,-(a7)      ; Zeiger in Datentabelle
1510        jsr      -126(pc)      ; ---> F813EC: Muster in BitPlane odern
1514        movea.l  d0,a2         ; a2 := d0 = Zeiger in Datentabelle
1516        lea      C(a7),a7      ; Stackzeiger korrigieren
151A        bra      F81496        ; ---> weiterlesen
151E
151E F8151E move.l   d6,-(a7)      ; Farbinfo
1520        move.l   d2,-(a7)      ; y2
```

```
1522        move.l  d3,-(a7)        ; x2
1524        move.l  d5,-(a7)        ; y1
1526        move.l  d4,-(a7)        ; x1
1528        jsr     F81564          ; ---> Strecke (x1,y1)-(x2,y2) zeichnen
152E        move.l  d3,d4           ; x1 := x2
1530        move.l  d2,d5           ; y1 := y2
1532        lea     14(a7),a7       ; Stackzeiger korrigieren
1536        bra     F81496          ; ---> weitermachen
153A
153A F8153A movem.l (a7)+,d2-d6/a2  ; Register wiederherstellen
153E        rts
1540
1540 ;----- Speicherbereich löschen
1540
1540 F81540 move.l  d2,-(a7)        ; d2 retten
1542        movea.l 8(a7),a0        ; a0 -> Anfang des Speicherbereichs
1546        move.l  C(a7),d1        ; d1 := Länge des Speicherbereichs
154A        move.w  12(a7),d2       ; d2 := Löschkode
154E        move.l  d1,d0           ; d0 := d1 = Länge des Bereichs
1550        asr.l   #1,d0           ; d0 := Länge in Worten
1552        move.l  d0,d1           ; d1 := d0 = Länge in Worten
1554 F81554 move.l  d1,d0           ; d0 := d1 = Restlänge in Worten
1556        subq.l  #1,d1           ; d1 dekrementieren
1558        move.l  d0,d0           ; Restlänge = 0?
155A        beq.s   F81560          ; Ja: fertig ->
155C        move.w  d2,(a0)+        ; Löschkode in Speicher schreiben
155E        bra.s   F81554          ; ---> wiederholen
1560
1560 F81560 move.l  (a7)+,d2        ; d2 wiederherstellen
1562        rts
1564
1564 ;----- Strecke zeichnen
1564
1564 F81564 movem.l d2-d4,-(a7)     ; Register retten
1568        move.l  10(a7),d4       ; d4 := x1
156C        move.l  14(a7),d3       ; d3 := y1
1570        move.l  18(a7),d2       ; d2 := x2
1574        move.l  d2,d1           ; d1 := d2 = x2
1576        sub.l   d4,d1           ; d1 := x2-x1
1578        move.l  1C(a7),d0       ; d0 := y2
```

```
157C        sub.l   d3,dØ              ; dØ := y2-y1
157E        tst.l   d1                 ; x2-x1 < Ø?
158Ø        blt.s   F81584             ; Ja: ->
1582        bra.s   F81586             ; --->
1584
1584 F81584 neg.l   d1                 ; x-Differenz positiv machen
1586 F81586 tst.l   dØ                 ; y2-y1 < Ø?
1588        blt.s   F8158C             ; Ja: ->
158A        bra.s   F8158E             ; --->
158C
158C F8158C neg.l   dØ                 ; y-Differenz positiv machen
158E F8158E cmp.l   dØ,d1              ; x-Differenz < y-Differenz?
159Ø        blt.s   F815AC             ; Ja: ->
1592        move.l  2Ø(a7),-(a7)       ; Farbinfo
1596        move.l  2Ø(a7),-(a7)       ; y2
159A        move.l  d2,-(a7)           ; x2
159C        move.l  d3,-(a7)           ; y1
159E        move.l  d4,-(a7)           ; x1
15AØ        jsr     F815CA             ; ---> Strecke zeichnen
15A6        lea     14(a7),a7          ; Stackzeiger korrigieren
15AA        bra.s   F815C4             ; ---> Ausgang
15AC
15AC F815AC move.l  2Ø(a7),-(a7)       ; Farbinfo
15BØ        move.l  2Ø(a7),-(a7)       ; y2
15B4        move.l  d2,-(a7)           ; x2
15B6        move.l  d3,-(a7)           ; y1
15B8        move.l  d4,-(a7)           ; x1
15BA        jsr     F816FE             ; ---> Strecke zeichnen
15CØ        lea     14(a7),a7          ; Stackzeiger korrigieren
15C4 F815C4 movem.l (a7)+,d2-d4        ; Register wiederherstellen
15C8        rts
15CA
15CA ;----- Strecke zeichnen (Betrag der Steigung < 1)
15CA
15CA F815CA movem.l d2-d6/a2-a4,-(a7)  ; Register retten
15CE        move.l  24(a7),d3          ; d3 := x1
15D2        move.l  28(a7),d2          ; d2 := y1
15D6        move.l  2C(a7),d1          ; d1 := x2
15DA        move.l  d1,dØ              ; dØ := d1 = x2
15DC        sub.l   d3,dØ              ; dØ := x2-x1
```

```
15DE         blt.s    F815E6          ; x2-x1 < Ø: positiv machen ->
15EØ         move.l   d1,dØ           ; dØ := d1 = x2
15E2         sub.l    d3,dØ           ; dØ := x2-x1
15E4         bra.s    F815EC          ; --->
15E6
15E6 F815E6  move.l   d1,dØ           ; dØ := d1 = x2
15E8         sub.l    d3,dØ           ; dØ := x2-x1 < Ø
15EA         neg.l    dØ              ; dØ := x1-x2 > Ø
15EC F815EC  move.l   dØ,d4           ; d4 := dØ = x-Differenz = Dx > Ø
15EE         move.l   3Ø(a7),dØ       ; dØ := y2
15F2         sub.l    d2,dØ           ; dØ := y2-y1
15F4         blt.s    F815FE          ; y2-y1 < Ø: ->
15F6         move.l   3Ø(a7),dØ       ; dØ := y2
15FA         sub.l    d2,dØ           ; dØ := y2-y1
15FC         bra.s    F816Ø6          ; --->
15FE
15FE F815FE  move.l   3Ø(a7),dØ       ; dØ := y2
16Ø2         sub.l    d2,dØ           ; dØ := y2-y1 < Ø
16Ø4         neg.l    dØ              ; dØ := y1-y2 > Ø
16Ø6 F816Ø6  move.l   dØ,d6           ; d6 := dØ = y-Differenz = Dy >= Ø
16Ø8         add.l    d6,d6           ; d6 := 2*Dy
16ØA         sub.l    d4,d6           ; d6 := 2*Dy-Dx
16ØC         movea.l  dØ,a4           ; a4 := dØ = Dy
16ØE         adda.l   a4,a4           ; a4 := 2*Dy
161Ø         movea.l  dØ,a2           ; a2 := dØ = Dy
1612         suba.l   d4,a2           ; a2 := Dy-Dx
1614         adda.l   a2,a2           ; a2 := 2*(Dy-Dx)
1616         moveq    #14,d5          ; d5 := 2Ø
1618         cmp.l    d1,d3           ; x2 > x1?
161A         ble.s    F81636          ; Ja: ->
161C         movea.l  d1,a3           ; a3 := d1 = x2
161E         move.l   3Ø(a7),d4       ; d4 := y2
1622         asl.l    #2,d4           ; d4 := 4*y2
1624         move.l   d4,dØ           ; dØ := d4 = 4*y2
1626         asl.l    #2,d4           ; d4 := 16*y2
1628         add.l    dØ,d4           ; d4 := 2Ø*y2
162A         cmp.l    3Ø(a7),d2       ; y2 > y1?
162E         bge.s    F81632          ; Nein: ->
163Ø         moveq    #EC,d5          ; d5 := -2Ø
1632 F81632  move.l   d3,d2           ; d2 := d3 = x1
```

```
1634         bra.s    F81662              ; ---> Punkte der Strecke zeichnen
1636
1636 F81636  movea.l d3,a3               ; a3 := d3 = x1
1638         move.l  d2,d4               ; d4 := d2 = y1
163A         asl.l   #2,d4               ; d4 := 4*y1
163C         move.l  d4,d0               ; d0 := d4 = 4*y1
163E         asl.l   #2,d4               ; d4 := 16*y1
1640         add.l   d0,d4               ; d4 := 20*y1
1642         cmp.l   30(a7),d2           ; y1 > y2?
1646         ble.s   F8164A              ; Nein: ->
1648         moveq   #EC,d5              ; d5 := -20
164A F8164A  move.l  d1,d2               ; d2 := d1 = x2
164C         bra.s   F81662              ; ---> Punkte der Strecke zeichnen
164E
164E F8164E  move.l  a3,d0               ; d0 := aktueller x-Wert
1650         addq.l  #1,a3               ; x-Wert um 1 erhöhen
1652         cmp.l   d0,d2               ; Endwert d2 erreicht?
1654         ble.s   F81676              ; Ja: fertig ->
1656         tst.l   d6                  ; d6 < 0?
1658         bge.s   F8165E              ; Nein: ->
165A         add.l   a4,d6               ; 2*Dy zu d6 addieren
165C         bra.s   F81662              ; --->
165E
165E F8165E  add.l   d5,d4               ; 20*y um 20 erhöhen bzw erniedrigen
1660         add.l   a2,d6               ; 2*(Dy-Dx) zu d6 addieren
1662 F81662  move.l  34(a7),-(a7)        ; Farbinfo
1666         move.l  d4,-(a7)            ; 20*y
1668         move.l  a3,-(a7)            ; x
166A         jsr     F8167C              ; ---> Pixel (x,y) setzen
1670         lea     C(a7),a7            ; Stackzeiger korrigieren
1674         bra.s   F8164E              ; ---> wiederholen
1676
1676 F81676  movem.l (a7)+,d2-d6/a2-a4   ; Register wiederherstellen
167A         rts
167C
167C ;-----  Pixel setzen
167C
167C F8167C  movem.l d2-d4,-(a7)         ; Register retten
1680         move.l  10(a7),d1           ; d1 := x
1684         move.l  14(a7),d3           ; d3 := 20*y
```

```
1688          move.l   18(a7),d2        ; d2 := Farbinfo
168C          move.l   d1,d0            ; d0 := d1 = x
168E          asr.l    #4,d0            ; d0 := x/16
1690          add.l    d3,d0            ; d0 := (x+320*y)/16
1692          asl.l    #1,d0            ; d0 := (x+320*y)/8
1694          move.w   d0,d4            ; d4 := d0
1696          move.l   #8000,d3         ; Bit 15 in d3 setzen
169C          move.b   d1,d0            ; d0 := d1 = x modulo 256
169E          andi.b   #F,d0            ; d0 := x modulo 16
16A2          asr.l    d0,d3            ; d3 um d0 Bits nach rechts schieben
16A4          btst.l   #0,d2            ; BitPlane 1 Farbwert = 0?
16A8          beq.s    F816BC           ; Ja: ->
16AA          movea.l  #4000,a0         ; a0 -> BitPlane 1
16B0          moveq    #0,d0            ; d0 := 0
16B2          move.w   d4,d0            ; d0 := (x+320*y)/8 = Offset
16B4          adda.l   d0,a0            ; a0 -> Wort, das Pixel (x,y) enthält
16B6          move.w   d3,d0            ; d0 := d3 = Pixelbit im Wort
16B8          or.w     d0,(a0)          ; Bit in BitPlane 1 setzen
16BA          bra.s    F816CE           ; --->
16BC
16BC F816BC   movea.l  #4000,a0         ; a0 -> BitPlane 1
16C2          moveq    #0,d0            ; d0 := 0
16C4          move.w   d4,d0            ; d0 := d4 = (x+320*y)/8 = Offset
16C6          adda.l   d0,a0            ; a0 -> Wort, das Pixel (x,y) enthält
16C8          move.w   d3,d0            ; d0 := d3 = Pixelbit im Wort
16CA          not.w    d0               ; Alle Bits in d0 flippen
16CC          and.w    d0,(a0)          ; Bit in BitPlane 1 löschen
16CE F816CE   btst.l   #1,d2            ; BitPlane 2 Farbwert = 0?
16D2          beq.s    F816E6           ; Ja: ->
16D4          movea.l  #6000,a0         ; a0 -> BitPlane 2
16DA          moveq    #0,d0            ; d0 := 0
16DC          move.w   d4,d0            ; d0 := d4 = (x+320*y)/8 = Offset
16DE          adda.l   d0,a0            ; a0 -> Wort, das Pixel (x,y) enthält
16E0          move.w   d3,d0            ; d0 := d3 = Pixelbit im Wort
16E2          or.w     d0,(a0)          ; Bit in Bitplane 2 setzen
16E4          bra.s    F816F8           ; ---> Ausgang
16E6
16E6 F816E6   movea.l  #6000,a0         ; a0 -> BitPlane 2
16EC          moveq    #0,d0            ; d0 := 0
16EE          move.w   d4,d0            ; d0 := d4 = (x+320*y)/8 = Offset
```

```
16F0        adda.l  d0,a0              ; a0 -> Wort, das Pixel (x,y) enthält
16F2        move.w  d3,d0              ; d0 := d3 = Pixelbit im Wort
16F4        not.w   d0                 ; Alle Bits in d3 flippen
16F6        and.w   d0,(a0)            ; Bit in BitPlane 2 löschen
16F8 F816F8 movem.l (a7)+,d2-d4        ; Register wiederherstellen
16FC        rts
16FE
16FE ;----- Strecke zeichnen (Betrag der Steigung >= 1)
16FE
16FE F816FE movem.l d2-d6/a2-a4,-(a7)  ; Register retten
1702        move.l  24(a7),d3          ; d3 := x1
1706        move.l  28(a7),d2          ; d2 := y1
170A        move.l  2C(a7),d1          ; d1 := x2
170E        move.l  d1,d0              ; d0 := d1 = x2
1710        sub.l   d3,d0              ; d0 := x2-x1
1712        blt.s   F8171A             ; x2-x1 < 0: positiv machen ->
1714        move.l  d1,d0              ; d0 := d1 = x2
1716        sub.l   d3,d0              ; d0 := x2-x1
1718        bra.s   F81720             ; --->
171A
171A F8171A move.l  d1,d0              ; d0 := d1 = x2
171C        sub.l   d3,d0              ; d0 := x2-x1 < 0
171E        neg.l   d0                 ; d0 := x1-x2 > 0
1720 F81720 move.l  d0,d4              ; d4 := d0 = x-Differenz = Dx > 0
1722        move.l  30(a7),d0          ; d0 := y2
1726        sub.l   d2,d0              ; d0 := y2-y1
1728        blt.s   F81732             ; y2-y1 < 0: positiv machen ->
172A        move.l  30(a7),d0          ; d0 := y2
172E        sub.l   d2,d0              ; d0 := y2-y1
1730        bra.s   F8173A             ; --->
1732
1732 F81732 move.l  30(a7),d0          ; d0 := y2
1736        sub.l   d2,d0              ; d0 := y2-y1 < 0
1738        neg.l   d0                 ; d0 := y1-y2 > 0
173A F8173A move.l  d4,d6              ; d6 := d4 = x-Differenz = Dx >= 0
173C        add.l   d6,d6              ; d6 := 2*Dx
173E        sub.l   d0,d6              ; d6 := 2*Dx-Dy
1740        movea.l d4,a4              ; a4 := d4 = Dx
1742        adda.l  a4,a4              ; a4 := 2*Dx
1744        movea.l d4,a2              ; a2 := d4 = Dx
```

```
1746         suba.l  d0,a2          ; a2 := Dx-Dy
1748         adda.l  a2,a2          ; a2 := 2*(Dx-Dy)
174A         moveq   #1,d5          ; d5 := 1
174C         cmp.l   30(a7),d2      ; y1 > y2?
1750         ble.s   F81770         ; Nein: ->
1752         movea.l d1,a3          ; a3 := d1 = x2
1754         move.l  30(a7),d4      ; d4 := y2
1758         asl.l   #2,d4          ; d4 := 4*y2
175A         move.l  d4,d0          ; d0 := d4 = 4*y2
175C         asl.l   #2,d4          ; d4 := 16*y2
175E         add.l   d0,d4          ; d4 := 20*y2
1760         cmp.l   d1,d3          ; x1 < x2?
1762         bge.s   F81766         ; Nein: ->
1764         moveq   #FF,d5         ; d5 := -1
1766 F81766  asl.l   #2,d2          ; d2 := 4*y1
1768         move.l  d2,d0          ; d0 := 4*y1
176A         asl.l   #2,d2          ; d2 := 16*y1
176C         add.l   d0,d2          ; d2 := 20*y1
176E         bra.s   F817A4         ; ---> Punkte der Strecke zeichnen
1770
1770 F81770  movea.l d3,a3          ; a3 := d3 = x1
1772         move.l  d2,d4          ; d4 := d2 = y1
1774         asl.l   #2,d4          ; d4 := 4*y1
1776         move.l  d4,d0          ; d0 := d4 = 4*y1
1778         asl.l   #2,d4          ; d4 := 16*y1
177A         add.l   d0,d4          ; d4 := 20*y1
177C         cmp.l   d1,d3          ; x1 > x2?
177E         ble.s   F81782         ; Nein: ->
1780         moveq   #FF,d5         ; d5 := -1
1782 F81782  move.l  30(a7),d2      ; d2 := y2
1786         asl.l   #2,d2          ; d2 := 4*y2
1788         move.l  d2,d0          ; d0 := d2 = 4*y2
178A         asl.l   #2,d2          ; d2 := 16*y2
178C         add.l   d0,d2          ; d2 := 20*y2
178E         bra.s   F817A4         ; ---> Punkte der Strecke zeichnen
1790
1790 F81790  cmp.l   d2,d4          ; Endwert d2 für 20*y erreicht?
1792         bge.s   F817B6         ; Ja: Fertig ->
1794         moveq   #14,d0         ; d0 := 20
1796         add.l   d0,d4          ; 20*y um 20 erhöhen
```

```
1798          tst.l   d6                  ; d6 < 0?
179A          bge.s   F817A0              ; Nein: ->
179C          add.l   a4,d6               ; 2*Dx zu d6 addieren
179E          bra.s   F817A4              ; --->
17A0
17A0 F817A0   adda.l  d5,a3               ; x um 1 erhöhen bzw. erniedrigen
17A2          add.l   a2,d6               ; 2*(Dx-Dy) zu d6 addieren
17A4 F817A4   move.l  34(a7),-(a7)        ; Farbinfo
17A8          move.l  d4,-(a7)            ; 20*y
17AA          move.l  a3,-(a7)            ; x
17AC          jsr     -132(pc)            ; ---> F8167C: Pixel (x,y) setzen
17B0          lea     C(a7),a7            ; Stackzeiger korrigieren
17B4          bra.s   F81790              ; ---> wiederholen
17B6
17B6 F817B6   movem.l (a7)+,d2-d6/a2-a4   ; Register wiederherstellen
17BA          rts
17BC
17BC ;-----   Fläche mit Farbe füllen
17BC
17BC F817BC   movem.l d2-d6/a2,-(a7)      ; Register retten
17C0          move.l  1C(a7),d4           ; d4 := x
17C4          move.l  20(a7),d5           ; d5 := 20*y
17C8          move.l  24(a7),d6           ; d6 := Untergrundfarbe
17CC          movea.l #F817BC,a2          ; a2 -> Routine für Rekursion
17D2          move.l  d5,-(a7)            ; 20*y
17D4          move.l  d4,-(a7)            ; x
17D6          jsr     F8183C              ; ---> Farbinfo des Pixels (x,y) lesen
17DC          cmp.l   d0,d6               ; Hat Pixel Untergrundfarbe?
17DE          addq.l  #8,a7               ; Stackzeiger korrigieren
17E0          bne.s   F81836              ; Pixel hat nicht Untergrundfarbe: ->
17E2          move.l  28(a7),-(a7)        ; Farbinfo für Ausfüllung
17E6          move.l  d5,-(a7)            ; 20*y
17E8          move.l  d4,-(a7)            ; x
17EA          jsr     -170(pc)            ; ---> F8167C: Pixel (x,y) setzen
17EE          move.l  34(a7),-(a7)        ; 20*y (Anfang)
17F2          move.l  d6,-(a7)            ; Farbinfo für Ausfüllung
17F4          move.l  d5,d3               ; d3 := d5 = 20*y
17F6          moveq   #14,d2              ; d2 := 20
17F8          add.l   d2,d3               ; d3 := 20*(y+1)
17FA          move.l  d3,-(a7)            ; 20*(y+1)
```

```
17FC        move.l  d4,-(a7)          ; x
17FE        jsr     (a2)              ; ---> Rekursion
1800        move.l  44(a7),-(a7)      ; 20*y (Anfang)
1804        move.l  d6,-(a7)          ; Farbinfo für Ausfüllung
1806        move.l  d5,-(a7)          ; 20*y
1808        move.l  d4,d3             ; d3 := d4 = x
180A        subq.l  #1,d3             ; d3 := x-1
180C        move.l  d3,-(a7)          ; x-1
180E        jsr     (a2)              ; ---> Rekursion
1810        move.l  54(a7),-(a7)      ; 20*y (Anfang)
1814        move.l  d6,-(a7)          ; Farbinfo für Ausfällung
1816        move.l  d5,d3             ; d3 := d5 = 20*y
1818        moveq   #14,d2            ; d2 := 20
181A        sub.l   d2,d3             ; d3 := 20*(y-1)
181C        move.l  d3,-(a7)          ; 20*(y-1)
181E        move.l  d4,-(a7)          ; x
1820        jsr     (a2)              ; ---> Rekursion
1822        move.l  64(a7),-(a7)      ; 20*y (Anfang)
1826        move.l  d6,-(a7)          ; Farbinfo für Ausfüllung
1828        move.l  d5,-(a7)          ; 20*y
182A        move.l  d4,d3             ; d3 := d4 = x
182C        addq.l  #1,d3             ; d3 := x+1
182E        move.l  d3,-(a7)          ; x+1
1830        jsr     (a2)              ; ---> Rekursion
1832        lea     4C(a7),a7         ; Stackzeiger korrigieren
1836 F81836 movem.l (a7)+,d2-d6/a2    ; Register wiederherstellen
183A        rts
183C
183C ;----- Farbinfo des Pixels (x,y) lesen
183C
183C F8183C movem.l d2-d3,-(a7)       ; Register retten
1840        move.l  C(a7),d0          ; d0 := x
1844        move.l  10(a7),d1         ; d1 := 20*y
1848        moveq   #0,d3             ; d3 := 0
184A        move.l  d0,d2             ; d2 := d0 = x
184C        asr.l   #4,d2             ; d2 := x/16
184E        add.l   d1,d2             ; d2 := (x+320*y)/16
1850        asl.l   #1,d2             ; d2 := (x+320*y)/8
1852        move.l  #8000,d1          ; Bit 15 in d1 setzen
1858        andi.b  #F,d0             ; d0 := x modulo 16
```

```
185C          asr.l    d0,d1              ; d1 um d0 Bits nach rechts schieben
185E          movea.l  #4000,a0           ; a0 -> BitPlane 1
1864          adda.l   d2,a0              ; a0 -> Wort, das Pixel (x,y) enthält
1866          moveq    #0,d0              ; d0 := 0
1868          move.w   (a0),d0            ; d0 := Pixelwort
186A          and.l    d1,d0              ; Pixelbit isolieren
186C          beq.s    F81872             ; Bit nicht gesetzt: ->
186E          moveq    #1,d0              ; Bit 0 setzen
1870          or.l     d0,d3              ; und in Farbinfo einodern
1872 F81872   movea.l  #6000,a0           ; a0 -> BitPlane 2
1878          adda.l   d2,a0              ; a0 -> Wort, das Pixel (x,y) enthält
187A          moveq    #0,d0              ; d0 := 0
187C          move.w   (a0),d0            ; d0 := Pixelwort
187E          and.l    d1,d0              ; Pixelbit isolieren
1880          beq.s    F81886             ; Bit nicht gesetzt: ->
1882          moveq    #2,d0              ; Bit 1 setzen
1884          or.l     d0,d3              ; und in Farbinfo einodern
1886 F81886   move.l   d3,d0              ; d0 := d3 = Farbinfo
1888          movem.l  (a7)+,d2-d3        ; Register wiederherstellen
188C          rts
188E
188E          DC.W     0
1890
1890 ;-----   Datenblock von Disk in Kickstart-RAM schreiben
1890
1890 F81890   movem.l  d2-d6/a2,-(a7)     ; Register retten
1894          movea.l  1C(a7),a2          ; a2 -> Stackrahmen
1898          move.l   20(a7),d5          ; d5 := Zieladresse
189C          move.w   26(a7),d2          ; d2 := logische Sektornummer
18A0          moveq    #0,d6              ; d6 := 0
18A2          moveq    #A,d3              ; d3 := 10 = Wiederholungszähler
18A4          moveq    #0,d0              ; d0 := 0
18A6          move.w   d2,d0              ; d0 := d2 = logische Sektornummer
18A8          divu     #B,d0              ; d0 durch 11 (Sektoren/Spur) dividieren
18AC          andi.l   #FFFF,d0           ; Rest der Division löschen
18B2          move.l   d0,d4              ; d4 := d0 = Spurnummer
18B4 F818B4   move.w   4(a2),d1           ; d1 := Nummer der Spur im Puffer
18B8          ext.l    d1                 ; auf Langwort erweitern
18BA          cmp.l    d4,d1              ; Gesuchter Sektor schon im Puffer?
18BC          bne      F818C8             ; Nein: ->
```

```
18CØ        btst.b  #6,1(a2)     ; Daten im Puffer ok?
18C6        bne.s   F818FA       ; Ja: Disk lesen nicht notwendig ->
18C8 F818C8 move.l  d4,-(a7)     ; Spurnummer übergeben
18CA        move.l  a2,-(a7)     ; Adresse des Stackrahmens übergeben
18CC        jsr     F8ØCCE       ; ---> Kopf auf die Spur setzen
18D2        move.l  a2,-(a7)     ; Adresse des Stackrahmens übergeben
18D4        jsr     F8ØDF4       ; ---> Spur von Disk lesen
18DA        move.l  dØ,d6        ; d6 := dØ = Fehlerkode
18DC        lea     C(a7),a7     ; Stackzeiger korrigieren
18EØ        bne     F81934       ; Fehler beim Lesen: ->
18E4        move.l  a2,-(a7)     ; Adresse des Stackrahmens übergeben
18E6        jsr     F8ØFA6       ; ---> Puffer aufbereiten
18EC        move.l  dØ,d6        ; d6 := dØ = Fehlerkode
18EE        addq.l  #4,a7        ; Stackzeiger korrigieren
18FØ        bne     F81934       ; Fehlerhafte Daten im Puffer: ->
18F4        bset.b  #6,1(a2)     ; Flag 'Daten in Ordnung' setzen
18FA F818FA movea.w d2,aØ        ; aØ := d2 = Sektornummer
18FC        move.w  d4,d2        ; d2 := d4 = Spurnummer
18FE        move.w  d2,d4        ; d4 := d2 = Spurnummer
19ØØ        add.w   d2,d2        ; d2 := 2*Spurnummer
19Ø2        move.w  d2,d3        ; d3 := d2 = 2*Spurnummer
19Ø4        asl.w   #2,d2        ; d2 := 8*Spurnummer
19Ø6        add.w   d3,d2        ; d2 := 1Ø*Spurnummer
19Ø8        add.w   d4,d2        ; d2 := 11*Spurnummer
19ØA        suba.w  d2,aØ        ; aØ := Nummer des Sektors in der Spur
19ØC        moveq   #Ø,d2        ; d2 := Ø
19ØE        move.w  aØ,d2        ; d2 := aØ = Sektornummer in der Spur
191Ø        move.l  d2,-(a7)     ; Sektornummer in der Spur übergeben
1912        move.l  a2,-(a7)     ; Adresse des Stackrahmens übergeben
1914        jsr     F81ØCE       ; ---> Adresse des Sektors ermitteln
191A        movea.l dØ,aØ        ; aØ := dØ = Adresse des Sektors
191C        pea     4Ø(aØ)       ; Anfang des Datenblocks im Sektor
192Ø        move.l  d5,-(a7)     ; Zieladresse übergeben
1922        pea     2ØØ          ; 512 = Anzahl Bytes im Datenblock
1926        jsr     F8ØF28       ; ---> Block an Zieladresse schreiben
192C        lea     14(a7),a7    ; Stackzeiger korrigieren
193Ø F8193Ø move.l  d6,dØ        ; dØ := d6 = Fehlerkode
1932        bra.s   F81956       ; ---> Ausgang
1934
1934 F81934 bclr.b  #6,1(a2)     ; Flag 'Daten in Ordnung' löschen
```

```
193A        subq.l  #1,d3              ; Wiederholungszähler erniedrigen
193C        ble.s   F8193Ø             ; Zähler abgelaufen: Fehlerausgang ->
193E        move.l  d3,d1              ; d1 := d3 = Zähler
194Ø        moveq   #3,dØ              ; dØ := 3
1942        and.l   dØ,d1              ; Bits Ø und 1 des Zählers isolieren
1944        bne     F818B4             ; nicht = Ø: Neuer Versuch ->
1948        move.l  a2,-(a7)           ; Adresse des Stackrahmens übergeben
194A        jsr     F8ØC4E             ; ---> Kopf auf Spur Ø setzen
195Ø        addq.l  #4,a7              ; Stackzeiger korrigieren
1952        bra     F818B4             ; ---> Neuer Versuch
1956
1956 F81956 movem.l (a7)+,d2-d6/a2     ; Register wiederherstellen
195A        rts
195C
195C ;----- BitPlane-Daten für Fehlergraphik
195C
195C F8195C DC.W    FFFF,3
196Ø        DC.W    5C,82,5C,71,9A,6,A7,6,E5,71,E5,82,5C,82
197C        DC.W    FFFF,3
198Ø        DC.W    64,7D,64,71,9D,D,A4,D,DD,71,DD,7D,64,7D
199C        DC.W    FFFE,Ø
19AØ        DC.W    5D,81
19A4        DC.W    FFFF,2
19A8        DC.W    9C,65,A5,65,AD,25,A6,1E,9B,1E,94,25,9C,65
19C4        DC.W    FFFE,Ø
19C8        DC.W    9D,64
19CC        DC.W    FFFF,2
19DØ        DC.W    9B,6F,9B,74,9E,77,A3,77,A6,74,A6,6F,A3,6C,9E,6C,9B,6F
19F4        DC.W    FFFE,Ø
19F8        DC.W    9E,74
19FC        DC.W    FFFF,FFFF
1AØØ
1AØØ ;----- BitPlane-Daten für Hand mit Kickstart-Disk
1AØØ
1AØØ F81AØØ DC.W    FFFF,1
1AØ4        DC.W    69,33,8Ø,33,8Ø,49,B7,49,B7,33,C3,33,CE,3E,CE,86
1A24        DC.W    C5,86,C5,6Ø,86,6Ø,84,5E,7B,5E,7A,6Ø,73,6Ø,73,69
1A44        DC.W    69,7Ø,69,33
1A4C        DC.W    FFFF,2
1A5Ø        DC.W    96,4A,96,5F
```

```
1A58        DC.W    FFFE,Ø
1A5C        DC.W    6B,6D
1A6Ø        DC.W    FFFE,Ø
1A64        DC.W    97,5F
1A68        DC.W    FFFF,1
1A6C        DC.W    67,7Ø,67,32,C4,32,DØ,3E,DØ,87,9C,87,9C,8C,98,94
1A8C        DC.W    94,99,9Ø,9C,8A,A5,82,A9,82,B4,5Ø,B4,5Ø,95,4F,95
1AAC        DC.W    4F,79,53,73,5A,6D,5B,69,5F,62,64,5F,67,5E,67,5E
1ACC        DC.W    64,6Ø,6Ø,62,5C,69,5B,6D,54,73,5Ø,79,5Ø,94,51,95
1AEC        DC.W    51,B3,6E,B3,6E,9E,76,9E,7A,9A,7A,87,78,84,78,7A
1BØC        DC.W    87,6D,87,61,84,5F,81,5F,84,62,84,69,83,6A,7C,6A
1B2C        DC.W    79,67,7Ø,6E,64,74,58,7D,58,7C,64,73,6Ø,72,5D,6F
1B4C        DC.W    6Ø,71,64,72,67,7Ø
1B58        DC.W    FFFF,1
1B5C        DC.W    78,65,7A,5E,82,5F,83,62,83,69,7C,69,78,65
1B78        DC.W    FFFF,1
1B7C        DC.W    79,84,79,7A,88,6D,88,61,C3,61,C3,86,7A,86,79,82
1B9C        DC.W    FFFF,1
1BAØ        DC.W    82,33,B5,33,B5,48,82,48,82,33
1BB4        DC.W    FFFF,1
1BB8        DC.W    A6,36,B1,36,B1,44,A6,44,A6,36
1BCC        DC.W    FFFF,3
1BDØ        DC.W    84,46
1BD4        DC.W    FFFE,Ø
1BD8        DC.W    84,47
1BDC        DC.W    FFFF,1
1BEØ        DC.W    A8,37,AF,37,AF,43,A8,43,A8,37
1BF4        DC.W    FFFF,2
1BF8        DC.W    A7,43
1BFC        DC.W    FFFE,Ø
1CØØ        DC.W    A9,42
1CØ4        DC.W    FFFF,1
1CØ8        DC.W    75,61,78,61,78,63,75,67,75,61
1C1C        DC.W    FFFF,1
1C2Ø        DC.W    6F,B3,6F,9F,76,9F,7B,9A,7B,91,7F,93,87,93,87,95
1C4Ø        DC.W    8B,9A,8F,9A,8F,9C,89,A5,81,A8,81,B3,6F,B3
1C5C        DC.W    FFFF,1
1C6Ø        DC.W    7B,87,7B,8C,8Ø,89,7B,87
1C7Ø        DC.W    FFFF,1
1C74        DC.W    7F,8A,7B,8C,7B,87,9Ø,87,86,91,85,91,87,8F,82,8A
```

```
1C94    DC.W    7F,8A
1C98    DC.W    FFFF,1
1C9C    DC.W    94,87,9B,87,9B,8C,97,94,94,98,8F,99,8C,99,89,95
1CBC    DC.W    89,92,94,87
1CC4    DC.W    FFFF,1
1CC8    DC.W    8A,92,8A,95,8C,98,8E,98,92,97,93,94,8F,91,8A,92
1CE8    DC.W    FFFF,1
1CEC    DC.W    7C,90,84,92,86,8F,82,8B,7F,8B,7C,8D,7C,90
1D08    DC.W    FFFF,1
1D0C    DC.W    C4,33,CF,3E,CF,86
1D18    DC.W    FFFE,0
1D1C    DC.W    68,33
1D20    DC.W    FFFE,0
1D24    DC.W    81,33
1D28    DC.W    FFFE,0
1D2C    DC.W    A7,37
1D30    DC.W    FFFE,0
1D34    DC.W    CF,43
1D38    DC.W    FFFE,0
1D3C    DC.W    B6,37
1D40    DC.W    FFFE,0
1D44    DC.W    CF,3E
1D48    DC.W    FFFE,0
1D4C    DC.W    C4,86
1D50    DC.W    FFFE,0
1D54    DC.W    91,88
1D58    DC.W    FFFE,0
1D5C    DC.W    74,61
1D60    DC.W    FFFC,5,8,9B7F       ; 'AMIGA' Muster 1
1D68    DC.W    1001,8220,4050,4,0
1D72    DC.W    404,8889,1210,5010,0
1D7C    DC.W    408,2448,1090,1020,0
1D86    DC.W    410,1042,510,1040,0
1D90    DC.W    420,50,2690,1080,0
1D9A    DC.W    440,484,900,1100,0
1DA4    DC.W    480,4960,4208,1200,0
1DAE    DC.W    500,683,8404,1400,0
1DB8    DC.W    FFFD,5,8,9B7F       ; 'AMIGA' Muster 2
1DC0    DC.W    1F9F,FE3F,FF77,FE7C,0
1DCA    DC.W    70C,EF8F,1E71,DC30,0
```

```
1DD4        DC.W    7F8,E7CE,1CF3,1FEØ,Ø
1DDE        DC.W    731,F3CE,1DF6,1CCØ,Ø
1DE8        DC.W    76Ø,3DC,3FFC,1D8Ø,Ø
1DF2        DC.W    7CØ,879C,3F78,1FØØ,Ø
1DFC        DC.W    78Ø,CF78,7E78,1EØØ,Ø
1EØ6        DC.W    7ØØ,FEFF,FC7C,1CØØ,Ø
1E1Ø        DC.W    FFFC,3,7,8F1Ø        ; 'Kick'
1E18        DC.W    C71F,1E63,8ØØØ
1E1E        DC.W    6631,8C63,Ø
1E24        DC.W    3EØ1,8C33,Ø
1E2A        DC.W    6631,8C1F,Ø
1E3Ø        DC.W    C61F,E33,Ø
1E36        DC.W    6ØØ,63,Ø
1E3C        DC.W    7ØØ,C63,8ØØØ
1E42        DC.W    FFFC,4,7,828C        ; 'start'
1E4A        DC.W    7Ø1E,6FØE,FCØ,Ø
1E52        DC.W    98ØC,319E,18ØØ,Ø
1E5A        DC.W    198C,3EØ3,F8Ø,Ø
1E62        DC.W    199C,3ØØ3,CØ,Ø
1E6A        DC.W    7CEE,1FØF,1F8Ø,Ø
1E72        DC.W    18ØØ,3,Ø,Ø
1E7A        DC.W    1ØØØ,2,Ø,Ø
1E82        DC.W    FFFF,FFFF
1E86
1E86 ********************* Ende von ROM Bootstrap ***************************
```

COMMODORE
AMIGA KNOW HOW

DR. RUPRECHT

KOMMENTIERTES ROM-LISTING

Der Autor Dr. Ruprecht gilt als einer der anerkanntesten Kenner der Commodore Betriebssysteme. Zu fast allen Commodore-Geräten hat er umfangreiche Betriebssystem-Dokumentationen erarbeitet und veröffentlicht.

Beim legendären CBM 8096 war er selbst maßgeblich an der Entwicklung des Betriebssystems LOS 96 beteiligt.

Byte für Byte hat der Autor die jeweiligen Stellen des Betriebssystems aufgelistet und mit Kommentaren versehen und damit das, was im Amiga-Betriebssystem abläuft, transparent und nachvollziehbar gemacht.

Dazu gehört auch die Beschreibung der in diesem Bereich abgelegten Routinen für Multitasking des Commodore Amiga.

Dieses vorliegende **ROM LISTING** behandelt den Betriebssystembereich der Kickstart-Ebene mit **EXEC, BOOT ROM und DOS-BOOT.**

Ein unentbehrliches Sachbuch und Nachschlagewerk für jeden, der sich professionell mit dem Amiga beschäftigt.

Bisher erschienen vom Autor:

Rom Listing CBM 3000	bei SM Software
Rom Listing CBM 8000	bei SM Software
Rom Listing CBM 8050	bei SM Software
Rom Listing CBM 8096	bei SM Software
Rom Listing +4 / C 16	bei Commodore
Rom Listing C 128	bei Markt & Technik

Eine Gemeinschaftsproduktion von:







Erschienen bei: Mediscript-Verlag · München

ISBN 3-88320-168-5

